12-2017

A Deep Learníng-Based Data Minimization Algorithm for Big Genomics Data in Support of IoT and Secure Smart Health Services

Mohammed Aledhari
Western Michigan University, aletharee@yahoo.com

A DEEP LEARNING-BASED DATA MINIMIZATION ALGORITHM FOR BIG GENOMICS DATA IN SUPPORT OF INTERNET OF THINGS (IOT) AND SECURE SMART HEALTH SERVICES

by

Mohammed Aledhari

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
Computer Science
Western Michigan University
December 2017

Doctoral Committee:

Fahad Saeed, Ph.D., Chair
Elise DeDoncker, Ph.D.
Leszek Lilien, Ph.D.
Todd Barkman, Ph.D.

A DEEP LEARNING-BASED DATA MINIMIZATION ALGORITHM FOR BIG GENOMICS DATA IN SUPPORT OF INTERNET OF THINGS (IOT) AND SECURE SMART HEALTH SERVICES

Mohammed Aledhari, Ph.D.

Western Michigan University, 2017

In the age of Big Genomics Data, institutes such as the National Human Genome Research Institute (NHGRI),1000-Genomes project, and the international cancer sequencing consortium are faced with the challenge of sharing large volumes of data between internationally dispersed sample collectors, data analyzers, and researchers, a process that up until now has been plagued by unreliable transfers and slow connection speeds. These occur due to the inherent throughput bottlenecks of traditional transfer technologies. One suggested solution is using the cloud as an infrastructure to solve the store and analysis challenges. However, the transfer and share of the genomics datasets between biological laboratories and from/to the cloud represents an ongoing bottleneck because of the amount of data, as well as the limitations of the network bandwidth. Therefore, transfer challenges can be solved by either increasing the bandwidth or minimizing the data size during the transfer phase.

One way to increase the efficiency of data transmission is to increase the bandwidth, which might not always be possible due to resource limitations. Another way to maximize channel capacity utilization is by decreasing the bits that need to be transmitted for a given dataset. Traditionally, transmission of big genomics datasets between two geographical locations is commonly done using general-purpose protocols, such as hypertext transfer protocol (HTTP) and file transfer protocol (FTP). In this dissertation, a novel deep learning-based data minimization algorithm is presented and aims to: 1) minimize the datasets during transfer over the carrier channels; 2) protect the data from the man-in-the-middle (MITM) and other attacks by changing the binary representation (codewords) several times for the same dataset.

This innovative data minimization strategy exploits the alphabet limitation of DNA sequences and modifies the binary representation (codewords) of dataset characters by using deep learning-based random sampling that utilizes the convolutional neural network (CNN) and Fourier transform theory. This algorithm ensures transmission of big genomics datasets with minimal bits and latency, thereby lending to a more efficient and expedient process. To evaluate this approach, extensive actual and simulated tests on various genomics datasets were conducted. Results indicate that the proposed data minimization algorithm is up to 99-fold faster and more secure than the current use of the HTTP data-encoding scheme and 96-fold faster than FTP on tested datasets.

ACKNOWLEDGEMENTS

First and foremost, praises and thanks to God, the Almighty, for His showers of blessings throughout my research work to complete the dissertation successfully. I wish to acknowledge my appreciation to certain people.

Deep appreciation and sincere gratitude to my dissertation advisor, Dr. Fahad Saeed, for giving me the opportunity to do research and providing invaluable guidance throughout this research. His expertise, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance.

Dr. Marianne Di Pierro, my supervisor, has been extremely helpful to me during my studies and writing publications. As my supervisor, Dr. Di Pierro worked closely with me during the research and dissertation writing periods. I am extremely grateful for what she has offered me. I would also like to thank her for her friendship, empathy, mentorship, and great sense of humor. Without Dr. Saeed and Dr. Di Pierro's help and encouragement this dissertation would not have been written or ever finished!

Thank you to Dr. Elise DeDoncker, Dr. Leszek Lilien, and Dr. Todd Barkman, the respective members of my committee for their wonderful support and valuable feedback. I am indebted to my fellow doctoral students, those who have moved on, those in the middle of their studies, and those just beginning for their support, feedback, and friendship. I am appreciative of the dedicated faculty and staff of the Department of Computer Science and the Graduate College of Western Michigan University for their assistance and support of my efforts and thank them wholeheartedly. I am extremely grateful to my parents for their love, prayers, caring and sacrifices and for educating and preparing me for my future. I wish to express my unqualified thanks to my wife and children for their love, prayers, patience, understanding, and continuing support to complete my dissertation. Last, but certainly not least, my thanks to all of those individuals who have supported me to complete the dissertation directly or indirectly.

Mohammed Aledhari

DEDICATION

To my mother and father, the sun and moon in my world and the reasons for my existence, your love and wisdom adorn my heart with rare brightness.

May God grant everlasting blessings to my parents, now in Heaven, their loving and peaceful home.

To my siblings, the bright stars in my life, everlasting symbols of love and fealty in this existence, and beyond.

To my professors, beacons of scientific knowledge who illuminate the evolutionary pathways with their guidance and mentorship.

To all of those who navigated the River with me, the tributaries who quenched my thirst and provided sustenance on this journey.

In gratitude and humility, I dedicate my work to you.

I was taught the way of progress is neither swift nor easy.

–Marie Curie

TABLE OF CONTENTS

Table of Contents—Continued

Table of Contents—Continued

Table of Contents—Continued

Table of Contents—Continued

LIST OF TABLES

List of Tables—Continued

LIST OF FIGURES

List of Figures—Continued

CHAPTER 1
INTRODUCTION

1.1 Background

DNA sequencing is needed in the most critical areas such as criminal investigations, genotyping and determination of disease-relevant genes or agents causing diseases, mutation analysis, screening of single nucleotide polymorphisms (SNPs), detection of chromosome abnormalities [1], global determination of post-translational modification [2], and to identify disease- and/or drug-associated genetic variants to advance precision medicine [3] [4]. Also, the use of the next-generation sequencing (NGS) technologies such as whole-genome sequencing (WGS) and whole-exome sequencing (WES), are significantly decrease the sequencing costs and enable the genomic datasets to join to the big data club.

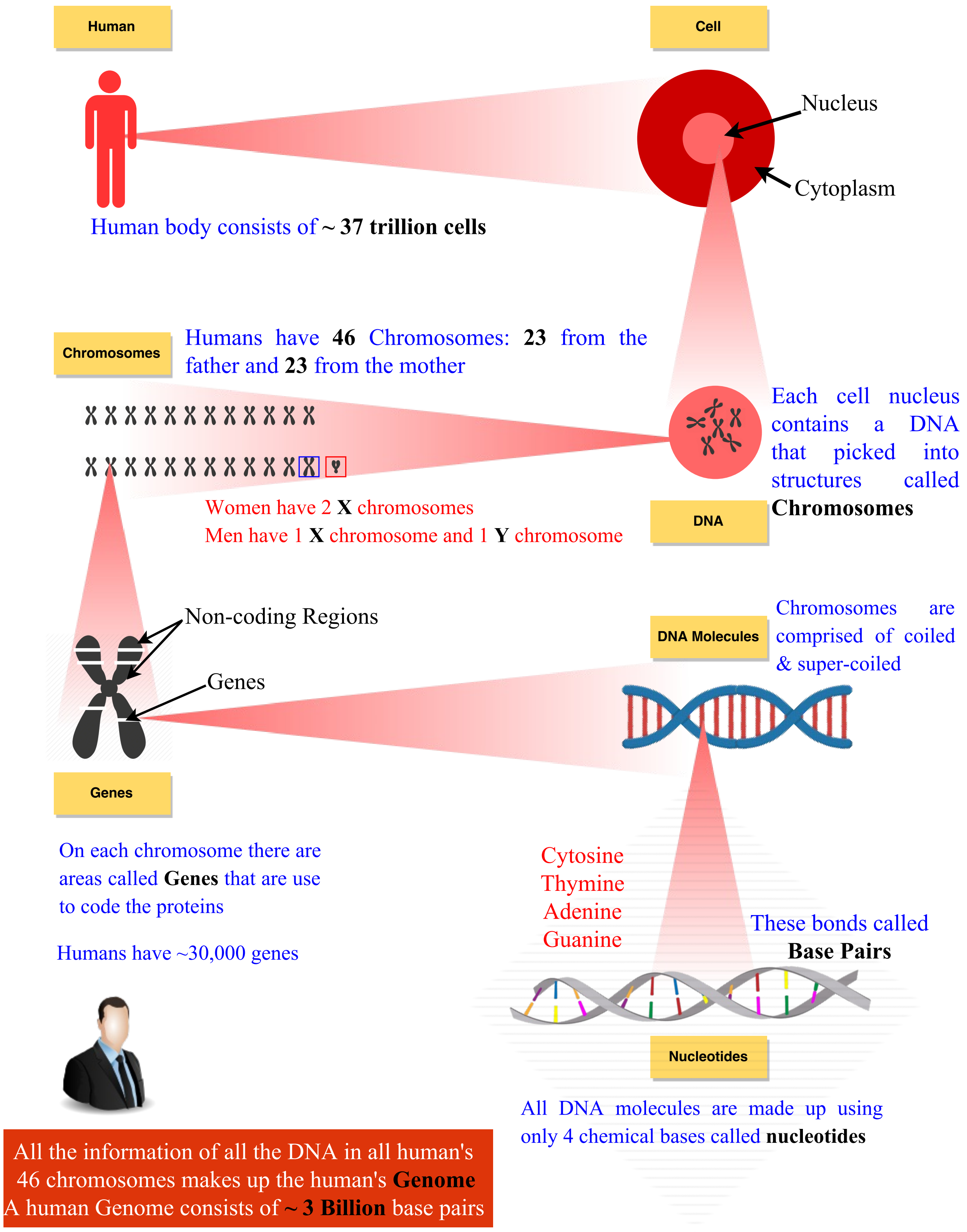
Currently, the major big data generators are Astronomy, YouTube, and Twitter and are expected to demonstrate continued dramatic growth in the volume of data to be acquired. For example, the Australian Square Kilometer Array Pathfinder (ASKAP) project currently acquires 7.5 terabytes/second of sample image data, a rate projected to increase 100-fold to 750 terabytes/second ($\sim$25 zettabytes per year) by 2025 [5] [6]. YouTube currently has 300 hours of video being uploaded every minute, and this could expand to 1,000 - 1,700 hours per minute (1 - 2 exabytes of video data per year) by 2025 if we extrapolate from current trends.

Today, Twitter generates 500 million tweets/day, each about 3 kilobytes including metadata. While this figure is beginning to plateau, a projected logarithmic growth rate would suggest a 2.4-fold growth by 2025, to 1.2 billion tweets per day, 1.36 petabytes/year. A big data generator will appear soon that will exceed 35 petabases per year [7]. For example, the cost of sequencing genomes reduced by a factor of 1 million in less than 10 years to reach to a few hundred dollars to sequence and map genomes faster than ever before. However, growing the genomic datasets brought challenges such as storing, handling, analyzing, visualizing, sharing, and transferring the genomic information generated by NGS technologies that need to be addressed. For instance, sequencing a single whole genome generates more than 250 gigabytes of data since there are over three billion base pairs (sites) on a human genome[8].

In fact, the growth rate of DNA sequencing over the last 10 years has generated a massive amount of data to produce a double amount approximately every 7 months. According to [9] to date, there are more than 2,500 high-throughput sequencing instruments distributed over 55 countries placed in about 1,000 sequencing centers.

The United States National Institutes of Health National Center for Biotechnology Information (NIH/NCBI) maintains the most archived sequencing reads. For example, NIH/NCBI currently maintains more than 3.6 petabases of sequence reads, distributed into approximately 32,000 microbial genomes, 5,000 animal and plant genomes, and 250,000 human genomes [10]. However, the current estimate of the global sequence reads is more than 35 petabases annually [11].

The tentative expectation of DNA sequencing for the next 8 years (2025) is one zettabase of annual sequencing and expanding to double per 7 months, as shown in Table 1.1 on page 10. There two more estimates of DNA sequencing that are doubling every 12 months according to Illumina's estimate [10] and every 18 months based on Moore's law. Also, biologists anticipate they need to sequence the most known species of plants and animals that are comprised of approximately 1.2 million genomes [1]. American and Chinese researchers plan to sequence about 1 million genomes in the next few years [12] [13]. For all listed information, it is necessary to prepare for the greatest challenge of big genomic datasets: data transfer. Recently, biologists ascertained that the bottleneck in the advent of the big genomics revolution is an inability to share and transfer large datasets in a timely manner. Therefore, some projects have begun to navigate the possible solutions to access big data and share them with researchers worldwide. The possible solutions are either to minimize the data volumes during the transfer over the networks or expand the network bandwidth. For example, the Human Genome Project [14] and the HapMap project



Human body consists of ~ **37 trillion cells**

Humans have **46** Chromosomes: **23** from the father and **23** from the mother

Each cell nucleus contains a DNA that picked into structures called **Chromosomes**

Women have 2 **X** chromosomes
Men have 1 **X** chromosome and 1 **Y** chromosome

Chromosomes are comprised of coiled & super-coiled

On each chromosome there are areas called **Genes** that are use to code the proteins

Humans have ~30,000 genes

These bonds called **Base Pairs**

All DNA molecules are made up using only 4 chemical bases called **nucleotides**

All the information of all the DNA in all human's 46 chromosomes makes up the human's **Genome**
A human Genome consists of ~ **3 Billion** base pairs

Figure 1.1: The human genome

[15] aim to facilitate sharing the sequence data and the more recent data-sharing structures for genome-wide association studies (GWAS) [16], such as dbGaP [17] and the European Genotyping Archive [18].

In addition, all of the large funding bodies now make data sharing a requirement of support for all projects, including all hypothesis-driven projects, whose primary purpose is to focus on a specific research question rather than to create data to be used by others. Implementing tools and techniques for accessing high-quality genomic datasets accelerates studies of the biological mechanisms of most diseases and the development of personalized treatments for individual patients. Individual researchers can no longer download and analyze the important datasets in their scientific fields on their own computers. Therefore, a solution to address this issue is necessary.

The purpose of this study is to remove the bottleneck of big genomic data access by implementing a novel data minimization algorithm to transfer data in a more expedient and secure way and to allow scientists to easily share their data and analyses as shown in Figures 1.2 on page 5 and 1.3 on page 6. In addition, data minimization solutions would support basic research and clinical trials by making data easily accessible, interoperable, and reusable.

1.2 Research Problem

Although low-cost, high-throughput instruments and cloud-based services have solved big data generating and processing challenges to certain extents, they do not solve the data transfer speed and security problems very efficiently when it comes to big genomic datasets. That is, transferring big data between two or more places (e.g. between two biology laboratories or between lab-cloud-lab) still results in a bottleneck due to the use of traditional transfer protocols such as HTTP [19] and FTP [20]. Recently, biologists ascertained that the bottleneck in the advent of the big genomics revolution results in an inability to share and transfer large datasets in a timely manner. Therefore, some projects have begun to navigate potential solutions to access big data and share them with researchers worldwide. The possible solutions are either to minimize the data volumes during the transfer over the networks [21] [22] or expand the network bandwidth [23]. For

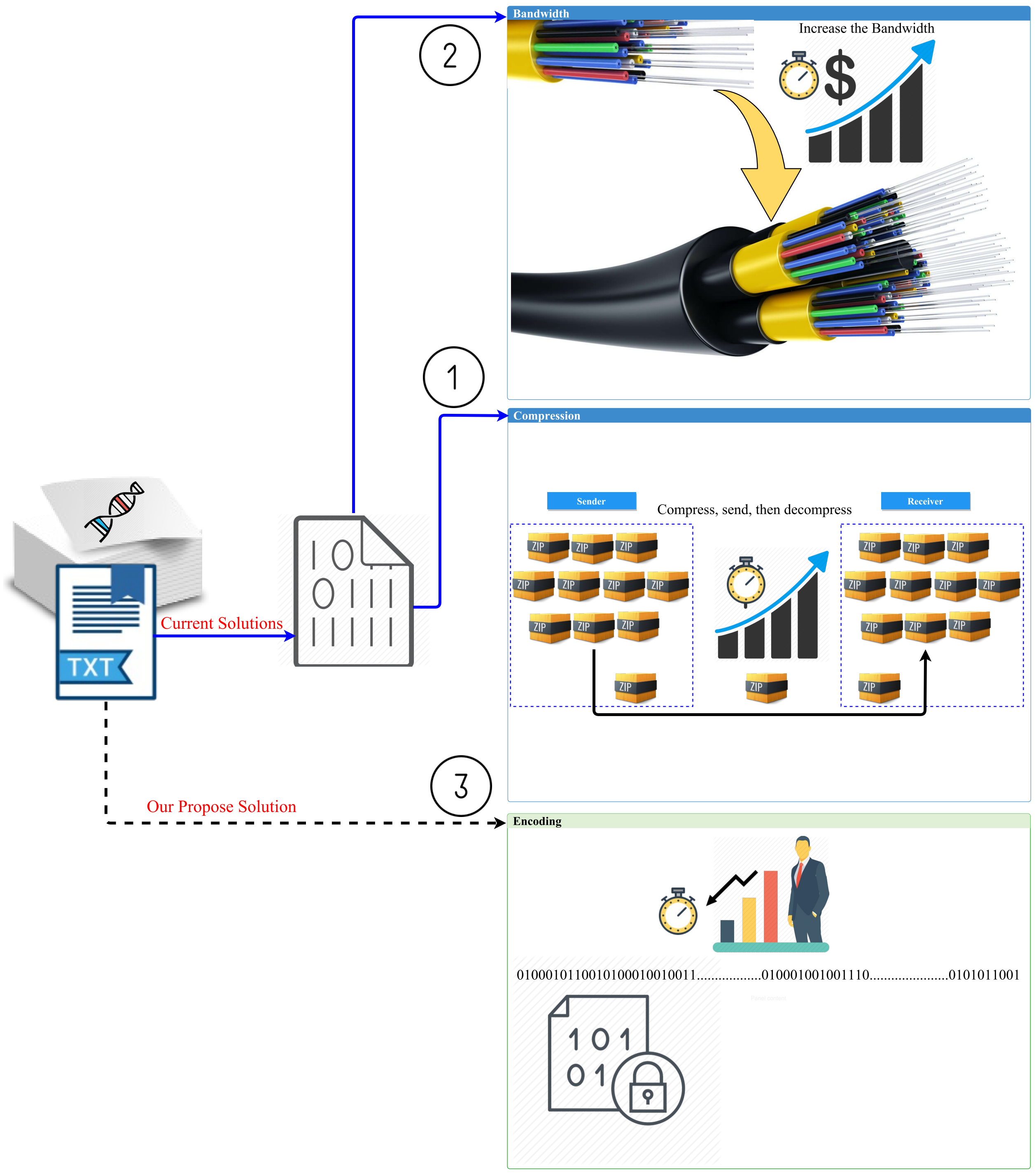


Figure 1.2: Comparison between available solutions and this work solution

example, the Human Genome Project [14] and the HapMap project [15] aims to facilitate sharing the sequence data and the more recent data-sharing structures for genome-wide association studies (GWAS) [16], such as dbGaP [17] and the European Genotyping Archive [18]. In addition, all of the large grantors and other funding agencies now make data sharing a requirement of support for all projects, including all hypothesis-driven projects, whose primary purpose is to focus on a specific research question rather than to create data to be used by others. Implementing tools and techniques for accessing high-quality genomic datasets accelerates studies of the biological mechanisms of most diseases and the development of personalized treatments for individual patients. Individual researchers can no longer download and analyze the important datasets in their scientific fields on their own computers, thereby creating an impasse in accessing critical information. A solution to address this issue is necessary.

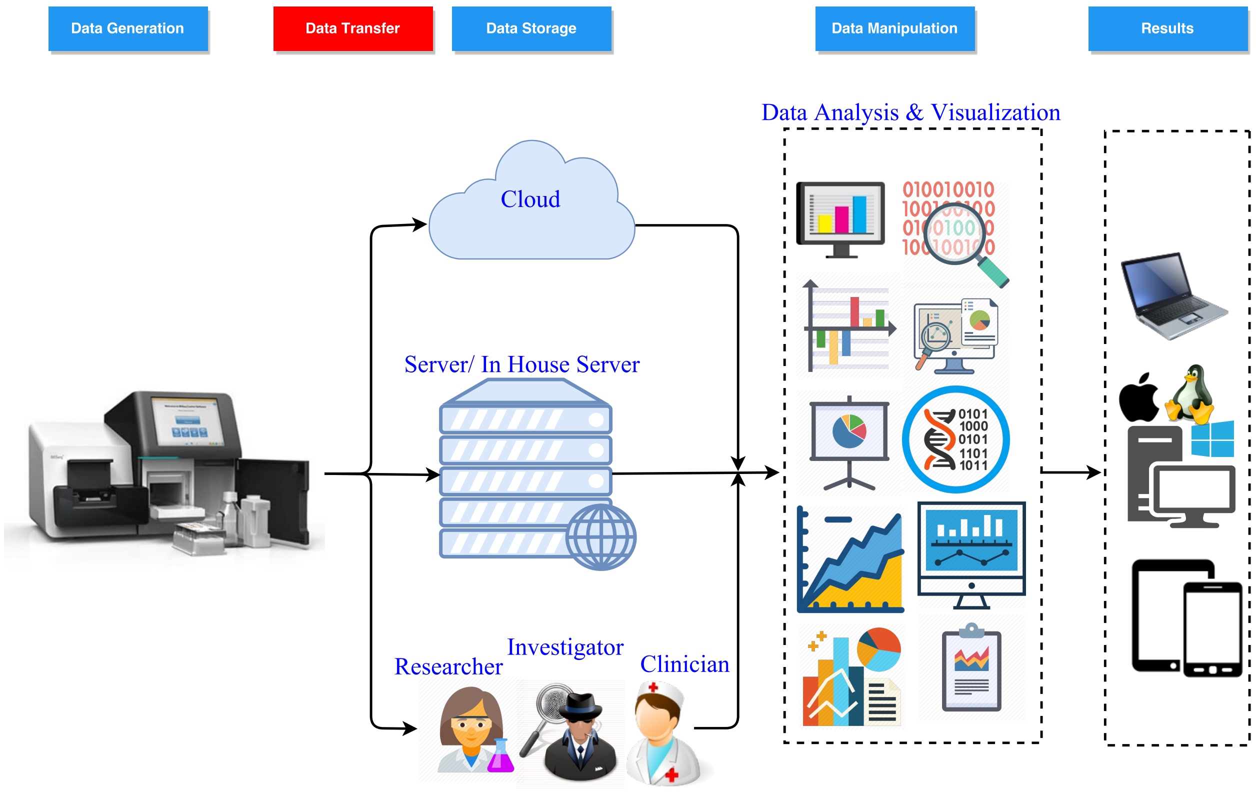


Figure 1.3: Genomic lifecycle

1.3 Purpose of the Study

We designed and implemented a novel data minimization algorithm to transfer big genomic datasets in a more expedient and secure way and to allow scientists to easily share their data and analyses. We used the HTTP as a baseline protocol [24] to compare and assess our implementation results of transferring big genomic datasets. The goals of our new data minimization algorithm are as follows: 1) reduce the size of data that need to be transferred between a server and a client [25]; 2) secure and protect the privacy of the data from unauthorized access due to attacks or data breach, such as MITM attack. Our heuristic model, simulation, and implementation results proved that our data minimization algorithm reduces significant amounts of data and makes more efficient use of network bandwidth, while also protecting the data by preventing unauthorized individuals from accessing them, should a breach occur. This dissertation represents an extension of our previous research dissertations in [26],[27], and [28], in which we changed the character codeword several times during transfer of a single dataset (file), a change that aims to minimize data transfers, thus shortening the overall transfer time of the genomic dataset and increasing data security. To the best of our knowledge, this is the first data minimization technique that reduces and secures the datasets during data transfer via changing binary representations of data characters many times for the same file. We show that our algorithm can provide remarkable improvements in response and transfer time of genomic datasets, as well as prevent unauthorized individuals from accessing the file contents in the event of a data breach: this occurs because we assign different codewords to the same character of the dataset in different times and file parts based on data obtained in running the convolutional neural network. We also illustrate the added benefit of using the deep learning technique of random sampling to form a renewable content-encoding in different times and file parts, an outcome that yields optimal results in terms of transfer data size, time, and security. Our approach is compatible with all existing browser implementations and specifications, such as Google Chrome [29], Safari [30], Internet Explorer [31], etc. Therefore, the overall benefit of this work is to increase opportunities for data sharing among researchers to advance the dissemination of scientific knowledge.

1.4 Audience

The audience for this work is researchers (biologists), investigators, and clinicians as shown in Figure 1.3 on page 6. Investigators use DNA sequencing to combat crimes by understanding finger printings and genetic clues in a crime scene: For example, the use of gel electrophoresis to relate sperm DNA to potential suspects. Also, investigators utilize DNA sequencing to understand ethnicity and ancestry by identifying ethnic characteristics of a certain country via specific patterns or genes. Thus, investigators can fully understand the development of mankind and its division throughout history. DNA sequencing also enables biologists and clinicians to investigate various diseases and genetic illnesses. In addition, many mutations are initiated by faulty genetic sequencing. Scientists can gain epidemiological data with multiple genomic candidates, and via genomic sequencing (in clinical trials), can provide critical information in the evolution of medical treatment.

1.5 Contribution

We create a new data minimization mechanism for big genomic datasets during real-time data transfer using a deep learning-based algorithm, as illustrated in section 3.3.3 on page 48. We assert that creating data minimization mechanisms to be equipped to transfer protocols such as HTTP and FTP can solve big data transmission challenges, especially for big genomic datasets in terms of transfer time and data security [32]. Our proposed data minimization mechanism for the transfer protocols enables them to be smart protocols via using standard codewords for dataset headers, while using our data minimization mechanism for the dataset body. We test our data minimization algorithm by using three different transfer protocols: HTTP, FTP and BitTorrent [33] and by considering such variables as versatility, security and flexibility. These are commonly used protocols that transfer different data types in a variety of browsers, such as Google Chrome. Also, these protocols have certain security features in the transport layer because they run on top of TCP [34]. These protocols are flexible because they are equipped with the ability to modify one or more components, such as content-encoding schemes, compression algorithms, and message headers. This dissertation is an extended version of our works published in [27][28]. We extend this previous work by employing the convolutional neural network to update the encoding codewords periodically and to ensure the assignment of the minimum binary representation to the most repetitive characters in the file.

1.6 Motivation

The revolution of the new technology solved the data generating challenges and resulted in the creation of big datasets. The generated data led to other challenges such as data storage, process, and share. Many efforts have been made to solve the challenges of big data storage, and manipulation, including data analysis and visualization. However, the challenges of big data sharing still constitute a major challenge that must be addressed and resolved. Also, designing and implementing transfer protocols equipped with data minimization techniques that relied on neural network techniques and that aimed to transfer big data in shorter times with added security against attacks, did not garner the attention of many researchers. Healthcare instruments and biology laboratories became big data generators that required singular methodologies in terms of transfer time, accuracy, speed, and security. Big genomic datasets are part of the big data club that require special handling from generating and processing to transferring between two or more biology laboratories. Many solutions have been developed to address the challenges of big data generating and analysis. However, transmission challenges have not been addressed at the same level, mainly due to compatibility issues. As a result, scientists are motivated to navigate and discover new mechanisms to transfer big genomic datasets more efficiently in terms of transfer time and security. Current content-encoding algorithms for transfer protocols use the standard encoding scheme [35], which increases the size and the transfer time, and which are not suitable for use with big genomic datasets. Observing these limitations, we take advantage of the nature of the genomic dataset alphabet to reduce the size of data being transferred. The genomic alphabet consists of only four characters; therefore, this alphabet yields a great reduction in data size and transfer time if encoded in smart ways such as using deep learning techniques during real-time data transfer. The term alphabet is defined as a symbol or group of symbols that can take different forms, such as alphabet of 2 symbols (bits) 0 and 1, of the 128 or 256 ASCII characters (8-bit), or any other characters.

Table 1.1: Four domains of big data in 2025

Data Lifecycle	Big Data Main Domains			
	Astronomy	**Twitter**	**YouTube**	**Genomics**
Acquisition	25 zetta bytes/year	0.5 -15 billion tweets/year	500 - 900 million hours/year	1 zetta bases/year
Storage	1 EB/year	1 - 17 PB/year	1 - 2 EB/year	2 - 40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, 2 trillion central processing unit (CPU) hours
	Massive volumes			All pairs genome alignments 10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern users bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

1.7 Dissertation Goals and Organization

The purpose of this dissertation is to develop and implement transfer protocols equipped with a novel data minimization algorithm for big genomic datasets that aim to share the data in less time and with more security. Moreover, these protocols will introduce a generic concept that can be modified to transfer securely minimum datasets that have limited symbols by using CNN-based algorithm content-encoding schemes. The implications of this dissertation are outlined as follows:

- Summarizes the standard and common use of content-encoding schemes that are currently employed in transfer protocols and their relevant standards to provide researchers with quick fundamentals, without having to search through the details presented in the standards' specifications.

- Provides an overview of some of the big genomic data challenges in terms of transmission and transfer time.

- Explores the relationship between big data and binary representation methods involving various binary encoding mechanisms.

- Presents the need for better transfer protocols equipped with data minimization algorithms to transfer datasets securely in shorter times, especially genomic datasets, and then to provide better services for big data demands.

- Implements, tests, and evaluates the proposed data minimization algorithm of transfer protocols in terms of transfer size, time, and security.

The remainder of this dissertation is organized as follows: Chapter 2 provides a literature review and summary of the related works that are used as a baseline for our implementations. Chapter 3 presents the first finding that is a deep learning-based data minimization algorithm for fast and secure transfer of big genomic datasets. Chapter 4 discusses the second finding that is a Fourier-based data minimization algorithm for fast and secure transfer of big genomic datasets, and the third and fourth findings which are variable-length and naive bit-based data minimization algorithm for fast and secure transfer of big genomic datasets in Chapter 5. Chapter 6 presents the fifth finding that is a new cryptography algorithm to protect cloud-based healthcare services presented. Finally, our conclusion is presented in Chapter 7.

CHAPTER 2
LITERATURE REVIEW

2.1 Work Related to Data Encoding Schemes

2.1.1 A Review of Related Network Transfer Protocols

Many protocols have been implemented to transfer different data volumes, such as HTTP and FTP, but these protocols operate in a data-oblivious manner. HTTP is a request/response protocol that resides in the first layer of the Open Systems Interconnection (OSI) conceptual model (application) and transfers data among web applications i.e. client(s)-server [36], as shown in Figure 2.1 on page 13.

HTTP communicates by sending a request from the client (requestor) to the server, followed by a response from the server to the client. Requests and responses are present in a simple ASCII format (8 bits). HTTP requests contain many elements: a method such as GET, PUT, POST and a uniform resource locater (URL). Also, HTTP requests include message headers and content-encoding, along with all data needed by the client. The server handles the request, and then responds according to the specified method. After that, the server sends a response to the client, including the status code, indicating if the request succeeded or failed and the reason why.

FTP is an application layer protocol of TCP/IP model that works on top of TCP and that transfers files between two machines only: i.e. client-server or machine-to-machine. FTP communicates by sending a request from the client to the server, along with a valid username and password. FTP needs two connection lines: one for commands called control connection and another one for data transfer called data connection. In FTP, data are encoded and can be compressed (optional) during the transfer phase, using a deflate compression algorithm via MODE Z command [37]. There are multiple binary codes available for FTP, such as a 7-bit, 8-bit, and 9-bit representation, which is considered to be problematic.

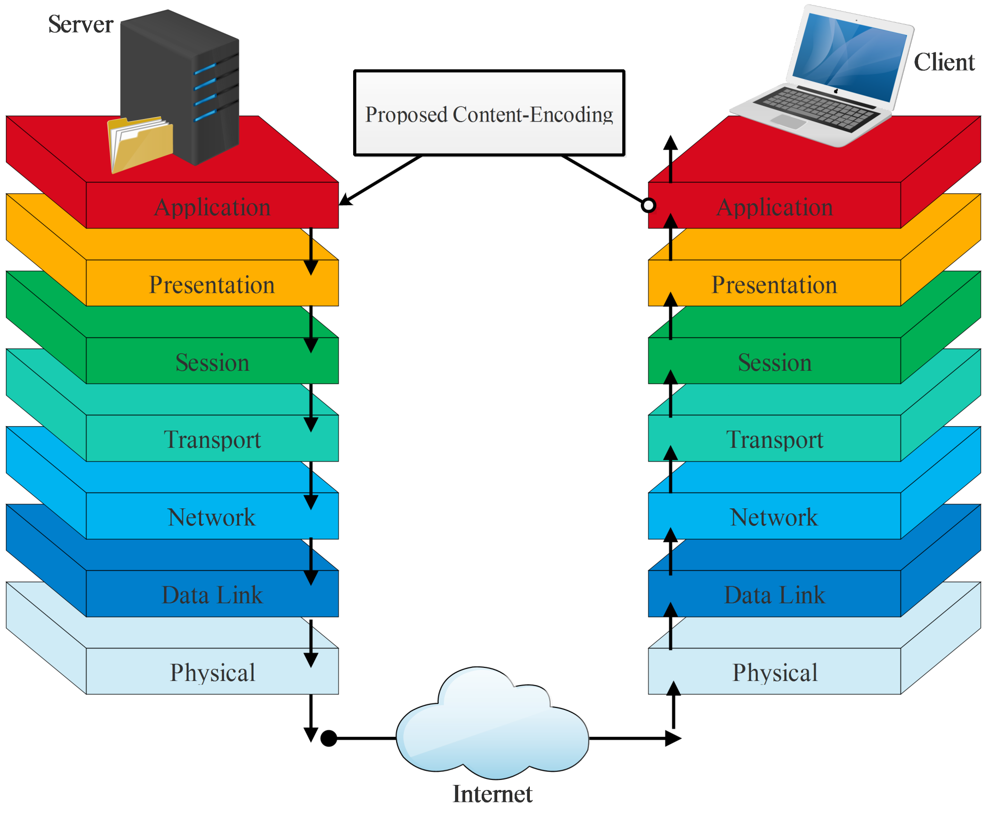


Figure 2.1: Open systems interconnection (OSI) conceptual model

We implement our content-encoding method for HTTP for the flexibility and security considerations that exist in HTTP, but not in FTP. Some of those flexibility considerations presented are needed to establish two consistent connections: command-line protocol (not friendly interface), and file size issue i.e. large file [38]. One of the major security concerns is the use of clear and unencrypted text, factors which make FTP easy to attack [39]. Two connections require more bandwidth, and that increases the cost, making it impractical. Also, if a disconnection happens for any reason during data transferring, there is a need to retransmit the whole file again [40]. HTTP-based protocol does not suffer from these drawbacks. This chapter introduces an adaptive binary encoding to speed up data transmission over HTTP.

2.1.2 A Review of Data Encoding Schemes

To the best of our knowledge, this work is the first network-based data minimization solution for big genomic datasets that utilizes data-encoding as a mechanism. GeneTorrent [41] is a file transfer protocol which uses the BitTorrent [42] technique to transfer genomic datasets, and which was originally designed to support distributed peer-to-peer (P2P) file transfer applications. In other words, GeneTorrent distributes the same file(s) on different machines settled in different locations and configures those machines to transfer certain part(s) of that file(s) to a requester. Although higher throughput can be achieved by using multiple machines for transferring data, the underlying data are still transferred using general-purpose protocols. This protocol is no longer in use, and there is a need to create a data-aware network transfer protocol for the DNA genomic datasets that use minimum resources of the network to deliver data efficiently.

The possible network solutions to transfer big genomic datasets expediently are listed as follows:

1. Enhance bandwidth utilization by developing new solutions and techniques for:
    - *Flow Control* [43]: An operation that balances the rate at which bits are generated by the sender with the rate at which bits are received by the receiver. This matches the speed of a sender with the capabilities of a receiver.
    - *Congestion Control* [44]: An operation that regulates the rate at which senders generate traffic in order to avoid the over-utilization of the resources available within network. This prevents network congestion which if pronounced, could lead to a network collapse.
2. Maximize the bandwidth by expanding the network in terms of physical (hardware) resources as such internet2 project[45] that provides high speed internet connection.
3. Minimize the datasets that can be achieved by developing new techniques and solutions for:

- Encoding schemes that deal with character codewords or binary representations, the focus and scope of this dissertation.

- Compression techniques that are out of this work scope. However, we list briefly the three different lossless compression algorithms: compress [46], deflate [47], and GZIP that can be used for HTTP as a preprocessing operation that requires additional time and continued research. Also, it would be worthy to mention here that there are some efficient compression algorithms that provide higher compression ratios, such as BZIP2 [48] and MFCompress, and that are genomic-specific compared to those that can be used in HTTP as preprocessing functions. However, the higher compression ratio algorithm is not the best choice when considering the compression time and security aspects of browsers. This tradeoff between compression ratio and compression time requires additional attention when dealing with time-sensitive applications, as shown later in the results. Also, some compression algorithms might suffer from security issues, such as intermediate proxies of the Chromium browser, which corrupts the data when trying to use BZIP2. Therefore, we introduce a new content-encoding that works best for all browsers, without adding more time or affecting security, such as those attached to compression algorithms. This work utilizes GZIP and MFCompress as benchmarks to compare with our encoding scheme over HTTP.

We can summarize some differences between the two data minimization techniques: encoding and compression, and then discuss why we decided to utilize data-encoding as a core of this work. These explanations are as follows:

1. Compression techniques cannot be implemented on the network during transfer phase, work in static environments such as workstations, PCs, and any non-transferable environments, which need network solutions.

2. Compression algorithms provide better performances in terms of data minimization but require longer time due to computation costs, which we want to avoid.

3. Compression algorithms use the same codeword for the entire dataset characters rather than all datasets, while we are able to change characters' codewords several times for each dataset to add extra security level via a proposed data-encoding scheme.

4. Not all compression techniques supported by network browsers such as Firefox, Edge, Safari, and etc.

5. Finally, the data-encoding techniques can provide similar performances to compression techniques in shorter transfer times and in more secure ways.

We put forward a novel network-based data minimization solution using CNN to transfer big genomic datasets expediently and securely, thereby enabling more scientists to share and analyze datasets.

2.1.3 Data-Encoding Approaches

Data minimization can be divided into two main forms: data encoding and data compression. Data minimization using data encoding assigns the lowest possible bits to each alphabet's symbol using content-encoding schemes without complex computations, whereas the data minimization using data compression assigns the lowest possible bits to the entire dataset: this process involves complex computations and an extended period of time to compress and decompress operations. In general, binary representation can be divided into two categories: Fixed-Length Binary Encoding (FLBE) and Variable-Length Binary Encoding (VLBE). FLBE scheme, also called singular encoding, converts symbols into a fixed number of output bits, such as in an ASCII code which consists of an 8-bit long for each codeword [49]. Variable-length binary encoding (VLBE), also referred to as a uniquely decodable and non-singular code, converts symbols into variable-length codewords, such that $\lambda_i \neq \lambda_j$ for all *i and j* [50]. However, the scope of this dissertation is data minimization using data-encoding techniques that are divided into five main mechanisms as follows:

2.1.4 Naive Bit Encoding

This approach works by assigning fixed-length codeword/binary representation to each alphabet symbol in a way that represents more than a single symbol in a single byte, such as 2-bit length to genomic symbols [51] and [52], as shown in Figure 2.2-(a) on page 18.

2.1.5 Dictionary-based/Substitutional Encoding

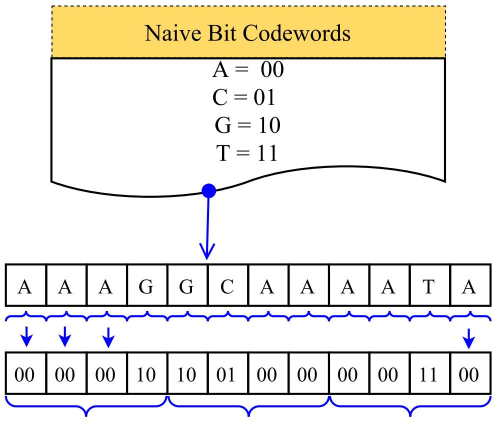
This approach stores different patterns of the input symbols in a dictionary or a database, along with their codewords, and then replaces the new input parts with predefined portions [53] and [54], such as in 1977-78 Ziv and Lempel (LZ-77) [55] and [56], as shown in Figure 2.2-(b) on page 18. LZ-77 algorithm works by replacing the repeated occurrences of symbols with their references that indicate length and location of that string, which occurred before, and which can be presented in the tuple (offset, length, symbol).

2.1.6 Statistical/Entropy Encoding

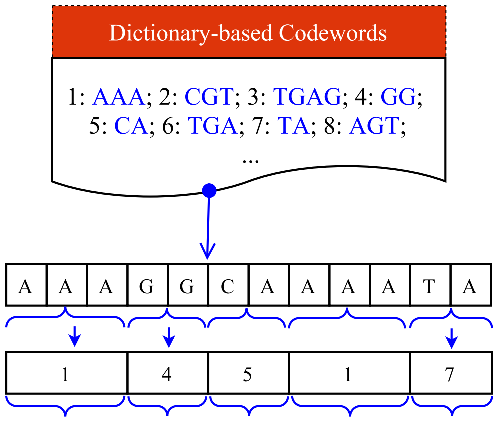
This approach works by statistics, prediction, and a probabilistic model from the input [57] and [58], such as Huffman's coding [55] and [59], as shown in Figure 2.2-(c) on page 18. Huffman's coding, introduced in 1952, is a statistical method that assigns a fixed-length codeword/binary representation to alphabet symbols, such as 2-bit, 3-bit, 8-bit, etc. The codewords will have different lengths, and the lowest frequency symbols will be assigned with the longest codewords and vice versa. This research utilizes this type of encoding with CNN deep learning algorithm to ensure the assignment of the lowest possible codewords to the more frequent dataset characters, and to undertake this process several times during the data transfer phase.

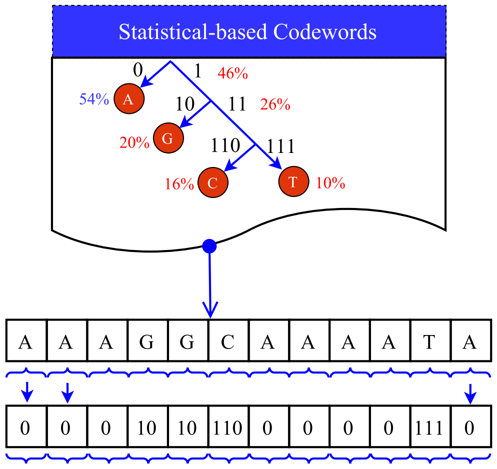
2.1.7 Referential/Reference-based Encoding

This approach is similar to a dictionary-based technique, except that it uses the pointer to the internal and external references, as shown in Figure 2.2-(d) on page 18.

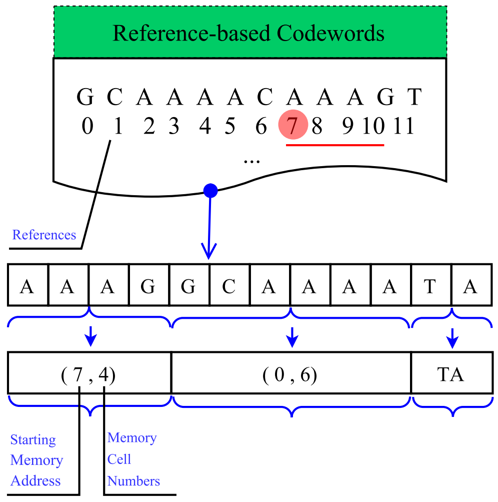


(a) Naive bit-based encoding


Dictionary-based Codewords
1: AAA; 2: CGT; 3: TGAG; 4: GG;
5: CA; 6: TGA; 7: TA; 8: AGT;
...
A A A G G C A A A A T A
1 4 5 1 7
(b) Dictionary-based encoding



(c) Statistical-based encoding

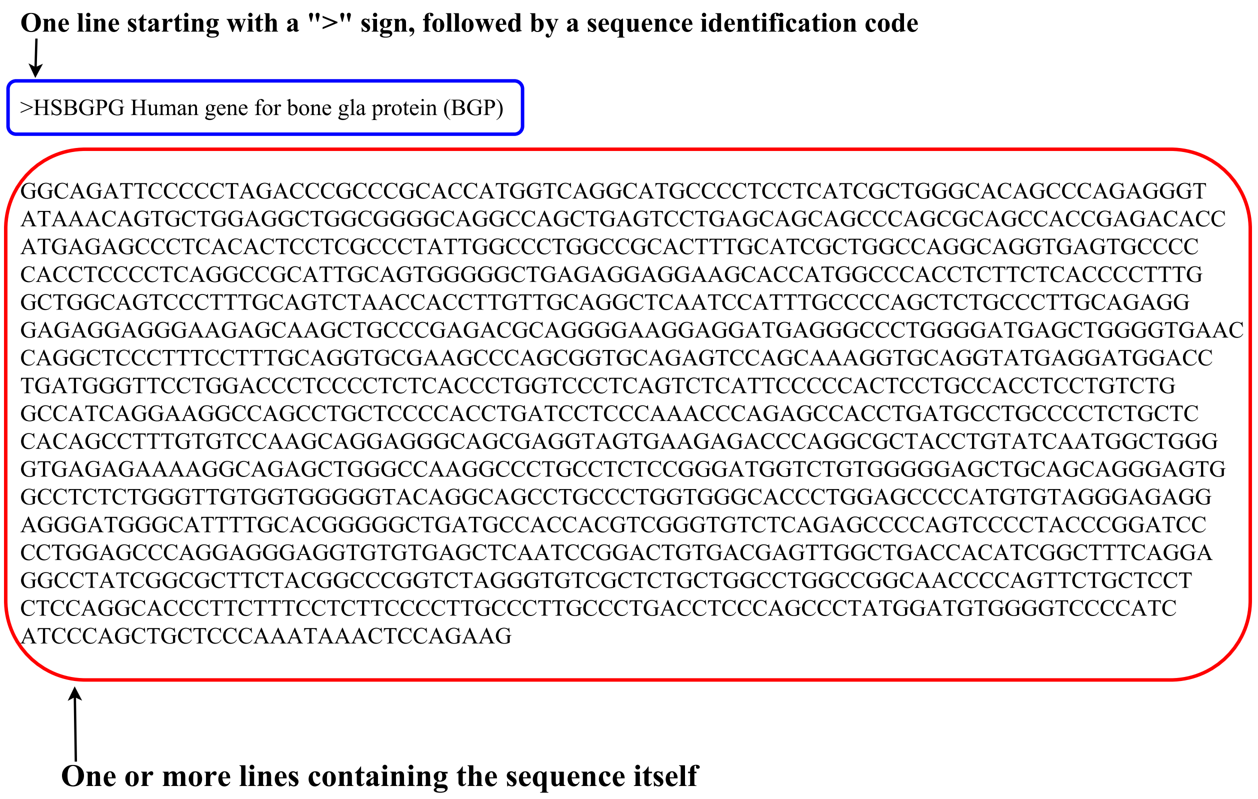


(d) Reference-based encoding

Figure 2.2: Data-encoding methods

2.1.8 Hybrid Encoding

This approach works by combining two or more encoding methods. For example, The Burrows-Wheeler transform (BWT) [60][55] and [61], is one of the hybrid encoding methods, especially popular in bioinformatics, used for data minimization. The BWT method works by permuting the input sequence in a way that symbols are grouped by their neighborhood. Our proposed data minimization algorithm can be classified as a hybrid encoding method by incorporating elements of the standard (8-bit) and the statistical encoding methods (variation of 1 - 3 bits.


One line starting with a ">" sign, followed by a sequence identification code

```
>HSBGPG Human gene for bone gla protein (BGP)
```

```
GGCAGATTCCCCCTAGACCCGCCCGCACCATGGTCAGGCATGCCCCTCCTCATCGCTGGGCACAGCCCAGAGGGT
ATAAACAGTGCTGGAGGCTGGCGGGGCAGGCCAGCTGAGTCCTGAGCAGCAGCCCAGCGCAGCCACCGAGACACC
ATGAGAGCCCTCACACTCCTCGCCCTATTGGCCCTGGCCGCACTTTGCATCGCTGGCCAGGCAGGTGAGTGCCCC
CACCTCCCCTCAGGCCGCATTGCAGTGGGGGCTGAGAGGAGGAAGCACCATGGCCCACCTCTTCTCACCCCTTTG
GCTGGCAGTCCCTTTGCAGTCTAACCACCTTGTTGCAGGCTCAATCCATTTGCCCCAGCTCTGCCCTTGCAGAGG
GAGAGGAGGGAAGAGCAAGCTGCCCGAGACGCAGGGGAAGGAGGATGAGGGCCCTGGGGATGAGCTGGGGTGAAC
CAGGCTCCCTTTCCTTTGCAGGTGCGAAGCCCAGCGGTGCAGAGTCCAGCAAAGGTGCAGGTATGAGGATGGACC
TGATGGGTTCCTGGACCCTCCCCTCTCACCCTGGTCCCTCAGTCTCATTCCCCCACTCCTGCCACCTCCTGTCTG
GCCATCAGGAAGGCCAGCCTGCTCCCCACCTGATCCTCCCAAACCCAGAGCCACCTGATGCCTGCCCCTCTGCTC
CACAGCCTTTGTGTCCAAGCAGGAGGGCAGCGAGGTAGTGAAGAGACCCAGGCGCTACCTGTATCAATGGCTGGG
GTGAGAGAAAAGGCAGAGCTGGGCCAAGGCCCTGCCTCTCCGGGATGGTCTGTGGGGGAGCTGCAGCAGGGAGTG
GCCTCTCTGGGTTGTGGTGGGGGTACAGGCAGCCTGCCCTGGTGGGCACCCTGGAGCCCCATGTGTAGGGAGAGG
AGGGATGGGCATTTTGCACGGGGGCTGATGCCACCACGTCGGGTGTCTCAGAGCCCCAGTCCCCTACCCGGATCC
CCTGGAGCCCAGGAGGGAGGTGTGTGAGCTCAATCCGGACTGTGACGAGTTGGCTGACCACATCGGCTTTCAGGA
GGCCTATCGGCGCTTCTACGGCCCGGTCTAGGGTGTCGCTCTGCTGGCCTGGCCGGCAACCCCAGTTCTGCTCCT
CTCCAGGCACCCTTCTTTCCTCTTCCCCTTGCCCTTGCCCTGACCTCCCAGCCCTATGGATGTGGGGTCCCCATC
ATCCCAGCTGCTCCCAAATAAACTCCAGAAG
```

One or more lines containing the sequence itself

Figure 2.3: The FASTA format components

2.2 Work Related to Data Encryption Methods

The security of personal health data is a critical issue when these data are transferred on wireless channels to end users such as doctors, nurses, family members, or other authorized individuals. The data are exchanged wirelessly where cables cannot be used, so all nodes can receive data if they are within range. If the network is not secure, an adversary could read, modify, and inject messages into the network. Such incorrect information, even when not for nefarious reasons, can lead to serious consequences for patients and potentially compromise their safe treatment. Current data encryption methods are not suitable for the cloud-based services such as remote healthcare monitoring due to using heterogeneous devices that use a variety of transfer protocols that belong to different vendors. Observing these facts, we take advantage of the nature of the genomic encryption and the deterministic Chaos Theory to implement a more efficient cryptography algorithm to secure remote healthcare monitoring.

When connecting the **W**ireless **S**ensor **N**etwork (WSN) to the Internet, the location of these sensor nodes would not be an important issue for intruders wherein they can attack the WSN from anywhere. Consequently, a powerful security mechanism should be designed with awareness of the resource constraints of the WSN. Most of the security protocols designed for WSN cannot be applied directly in the **W**earable **W**ireless **B**ody **A**rea **N**etwork (WWBASN), since these nodes have limited resources in the power, computation processing, and communication. A powerful method of data encryption is the one-time-pad algorithm [62], where each single piece of data is encrypted individually with a unique key. The disadvantage of this method is that it requires a vast number of keys; a **P**seudo **R**andom **N**umber **G**enerator (PRNG) could be used to generate the required keys, but it is problematic in terms of the key repetition. To eliminate the problem of repetition, the application of Chaos Theory to generate these keys represents a solid approach. The security algorithm of these networks must maintain the resource limitations of the sensor nodes and at the same time provide high security. The new generation of security mechanisms is genomics encryption due to its randomness and complexity.

The security algorithm of these networks must maintain the resource limitations of the sensor nodes and at the same time provide high security. As a method of authentication, Wang et al., [63] used a wavelet domain Hidden Markov Model. In addition, information from the **E**lectro**C**ardio**G**raphy (ECG) signal is used as a biometric key. Venkatasubramanian et al., [64] had proposed a **P**hysiological **S**ignal-based **K**ey **A**greement (PSKA) which shares the cryptographic key using physiological signals obtained from the patient. Liu et al., [65] had proposed a simulation of a chaotic block cipher used for wireless sensor networks through which results were compared with RC5 and RC6 block ciphers. An improvement of a **M**essage **A**uthentication **C**ode (MAC) algorithm was proposed using chaos and XOR encryption [66]. The advanced MAC generator has been divided into a sub key generator and the MAC structure. Genomic-based cryptography emerged as a new cryptographic field, in which nucleotide is used as an information carrier, and modern biological technology is used as an implementation tool. Adleman et al., [67] had solved **H**amiltonian **P**ath **P**roblems (HPP) by using nucleotide computing, with its inherent advantages, such as vast parallelism and extraordinary information density. Zhang et al., [68] had designed a genomics and chaos map to encrypt images. Karakose and Cigdem [69] had introduced a new approach to the genomics computing algorithm aims to perform genomics computing with adaptive parameters by using **Q**uantum-behaved **P**article **S**warm **O**ptimization (QPSO). Mokhtar et al., [70] had designed an RGB image encryption algorithm using DNA encoding and a chaos map.

In 2015, UbaidurRahman et al., [71] had designed a DNA-based encryption and decryption algorithm such that it overcame the limitation of the genomics-based cryptography algorithm and incorporated modular arithmetic cryptography at some steps. Also, in 2015 Gritti et al., [72] had designed a system that realizes complex access control on encrypted data, where the attributes used to describe a user's credentials will determine a policy as to which recipient will be able to decrypt the provided cipher text. Moosavi et al., [73] developed a secure authentication and authorization for patient security and privacy medical data for the Internet of Things (IoT)-based healthcare monitoring system. Farash et al., [74] proposed an improvement to the **U**ser **A**uthentication and **K**ey **A**greement **S**cheme (UAKAS) for **H**eterogeneous WSN (HWSN) by improving the security level and enabling the HWSN to dynamically grow, without influencing any part involved in the UAKAS.

CHAPTER 3

A DEEP LEARNING-BASED DATA MINIMIZATION ALGORITHM FOR FAST AND SECURE TRANSFER OF BIG GENOMIC DATASETS

In the age of Big Genomics Data, institutions such as the National Human Genome Research Institute (NHGRI) are faced with the challenge of sharing large volumes of data between internationally dispersed sample collectors, data analyzers, and researchers, a process that up until now has been plagued by unreliable transfers and slow connection speeds. These occur due to the inherent throughput bottlenecks of traditional transfer technologies. Two factors that affect the efficiency of data transmission are the channel bandwidth and the amount of data. One way to increase the efficiency of data transmission is to increase the bandwidth, which might not always be possible due to resource limitations. Another way to maximize channel utilization is by decreasing the bits that need to be transmitted for a given dataset.

Traditionally, transmission of big genomic data between two geographical locations is commonly done using general-purpose protocols, such as hypertext transfer protocol (HTTP) and file transfer protocol (FTP) secure. In this chapter, we present a novel machine learning-based data minimization algorithm that aims to: 1) minimize the datasets during transfer over the carrier channels; 2) protect the data from the man-in-the-middle (MITM) attack and other attacks by changing the binary representation (content-encoding) several times for the same dataset. In other words, assigning different codewords to the same character in different parts of the dataset. Our data minimization strategy exploits the alphabet limitation of DNA sequences and modifies the binary representation (codeword) of dataset characters using machine learning-based random sampling and convolutional neural network (CNN) techniques to assure that the minimum code word uses to the high frequency characters at different time slots of the transfer time.

This algorithm ensures transmission of big genomic DNA datasets with minimal bits and latency, thereby leading to a more efficient and expedient process. To evaluate our approach, we conducted extensive tests on various genomic datasets (actual and simulated). Our heuristic model, simulation, and real implementation results indicate that the proposed data minimization algorithm is up to 99 times faster and more secure than the currently used content-encoding scheme used in HTTP of the HTTP content-encoding scheme and 96 times faster than FTP on tested datasets. The developed protocol in C# will be available to the wider genomics community and domain scientists.

3.1 Introduction

High-throughput DNA sequencing instruments, such as next-generation sequencing (NGS) machines, have dropped sequencing prices significantly [75]. Those instruments became big data generators, not only for big biology centers, but also for small biology laboratories and researchers, as shown in Figure 3.1. Biology laboratories, even without NGS, are accessing terabytes of DNA sequences from public genomic repositories [76]. Although low-cost high-throughput instruments and cloud-based services have solved big data generating and processing challenges to some extent, they do not solve the data transfer speed and security problems very efficiently when it comes to big genomic datasets. That is, transferring big data between two or more places (e.g. between two biology laboratories or between lab-cloud-lab) still results in a bottleneck due to the use of traditional transfer protocols such as HTTP [19] and FTP [37]. Moreover, current data transfer protocols do not consider data minimization, multiple modifications to the character codewords for the same dataset to increase security, and fully utilize available bandwidth. This is partially due to not considering the data type, scope, and encoding schemes that cause unpredictable and slow transfers.

The current transfer protocols, such as HTTP or FTP content-encoding, use between an 8-16 bits long codeword for each alphabet character. Improving big data transfer performance can be accomplished via three main techniques: data parallelism; [77] [78], increase of bandwidth; [23]; and data minimization [21] [22]. In addition, considering the protocols available to transfer data, including big data, such as HTTP and FTP.

We designed and implemented a new content-encoding for big genomic data transfer that aims to minimize and secure the datasets during data transfer. We used the standard HTTP content-encoding scheme as a baseline protocol [24] to compare and assess our implementation results of transferring big genomic datasets. The goals of our new data minimization algorithm are as follows: 1) reduce the size of data that need to be transferred between a server and a client [25]; 2) secure and protect the privacy of the data from unauthorized access due to attacks or data breach, such as MITM attack. Our heuristic model, simulation, and implementation results proved that our data minimization algorithm reduces significant amounts of data and makes more efficient use of network bandwidth while protecting the data by preventing unauthorized individuals from accessing them, should a breach occur.

This chapter represents an extension of our previous published research papers in [26], [79], and [28], in which we changed the character codeword several times during transfer of a single dataset (file), a change that aims to minimize data transfers, thus shortening the overall transfer time of the genomic dataset and increasing data security. To the best of our knowledge, this is the first data minimization technique that aims to reduce and secure the datasets during data transfer via changing binary representations of data characters many times for the same file. We showed that our algorithm can provide remarkable improvements in response and transfer time of genomic datasets, as well as prevents unauthorized individuals from accessing the file contents in case of a data breach: this occurs because we assigned different codewords to the same character of the dataset in different times and file parts based on data obtained of running the convolutional neural network. We also illustrate the added benefit of using the machine learning technique of random sampling to form a renewable content-encoding in different times and file parts, an outcome that yields optimal results in terms of transfer data size, time, and security. Our approach is compatible with all existing browser implementations and specifications, such as Google Chrome [29], Safari [30], Internet Explorer [31], etc.

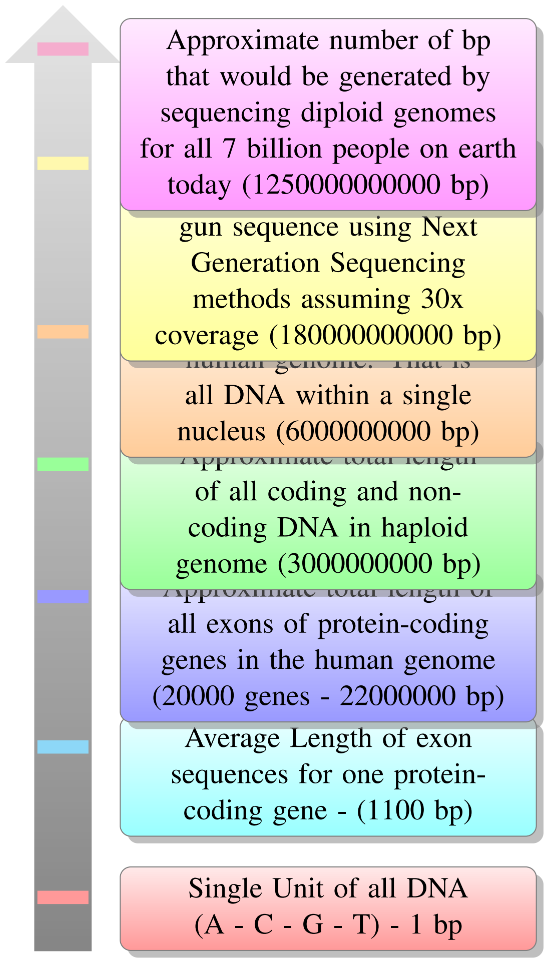

Approximate number of bp that would be generated by sequencing diploid genomes for all 7 billion people on earth today (1250000000000 bp)
gun sequence using Next Generation Sequencing methods assuming 30x coverage (180000000000 bp)
all DNA within a single nucleus (6000000000 bp)
of all coding and non-coding DNA in haploid genome (3000000000 bp)
all exons of protein-coding genes in the human genome (20000 genes - 22000000 bp)
Average Length of exon sequences for one protein-coding gene - (1100 bp)
Single Unit of all DNA (A - C - G - T) - 1 bp

Figure 3.1: The estimate digitalization of the world's population genomes on June 2016 [80] in a binary form

3.1.1 Contribution

Creating a new data minimization mechanism for big genomic datasets during real-time data transfer using a machine learning-based random sampling technique, as illustrated in section 3.3.3 on page 48. We assert that creating data minimization mechanisms to be equipped to transfer protocols such as HTTP and FTP can solve big data transmission challenges, especially for big genomic datasets in terms of transfer time and data security [32]. Our proposed data minimization mechanism for the transfer protocols enables them to be smart protocols via using standard codewords for dataset headers, while using our data minimization mechanism for the dataset body. We test our data minimization algorithm using three different transfer protocols that are HTTP, FTP and BitTorrent [33] for several considerations, such as versatility, security and flexibility. These are commonly used protocols that transfer different data types in a variety of browsers, such as Google Chrome. Also, these protocols have some kind of security features in the transport layer because they run on top of TCP [34]. These protocols are flexible because they are equipped with the ability to modify one or more components, such as content-encoding schemes, compression algorithms, and message headers.

This chapter is an extended version of our works published in data [79] and [28]. We extend this previous work by employing the convolutional neural network to update the encoding codewords periodically that ensures assigning the minimum binary representation to the most repetitive characters in the file.

3.1.2 Motivation

The revolution of the new technology nowadays solved the data generating challenges, result big datasets. The generated data assisted in forming other challenges such data store, process, and share. Many efforts have been made aiming to solve the big data store and manipulate challenges, including data analysis and visualization. However, the challenges of big data sharing still a major challenge that need to be addressed and solved. Also, designing and implementing transfer protocols equipped with data minimization techniques that relies on neural network techniques and aim to transfer the big data in shorter times and more security against attacks, did not bring attention of many researchers yet.

Healthcare instruments and biology laboratories became one of the big data generators that need to be treated in unusual ways in terms of transfer time, accuracy, speed, and security. Big genomic datasets are part of big data club that require special handle from generating and processing to transferring between two or more biology laboratories. Many solutions have been developed for challenges of big data generating and analysis. However, transmission challenges have not been tackled at the same level, mainly due to compatibility issues. As a result, scientists are motivated to navigate and find new mechanisms to transfer big genomic datasets more efficiently in terms of transfer time and security.

Current transfer protocols' content-encoding algorithms use the standard encoding scheme [35], which add extra size and transfer time, and which are not suitable for use to big genomic datasets. Observing these limitations, we take advantage of the nature of genomic dataset alphabet to reduce the size of data being transferred. The genomic alphabet consists of only four characters, therefore thais alphabet can yield a great reduction in data size and transfer time, if encoded in smart ways like using deep learning techniques during the real-time data transferring. The term alphabet is defined as a symbol or group of symbols that can take different forms, such as alphabet of 2 symbols (bits) 0 and 1, of the 128 or 256 ASCII characters (8-bit), or any other characters.

3.1.3 Chapter Goals and Organization

The purpose of this chapter is to develop and implement transfer protocols equipped with a novel data minimization algorithm for big genomic datasets aiming to share the data in reduced time and with more security. Moreover, it will introduce a generic concept that can be modified to transfer securely minimum datasets that have limited symbols by using machine learning-based random sampling content-encoding schemes. The contributions of this chapter are outlined as follows:

- Summarizes the standard and common use content-encoding schemes that are currently employed in transfer protocols and their relevant standards to provide researchers with quick fundamentals, without having to search through the details presented in the standards' specifications.
- Provides an overview of some of the big genomic data challenges in terms of transmission and transfer time.
- Explores the relationship between big data and binary representation methods involving various binary encoding mechanisms.

- Presents the need for better transfer protocols equipped with data minimization algorithms to transfer datasets securely in shorter times, especially genomic datasets, and then to provide better services for big data demands.

- Implement, test, and evaluate the proposed data minimization algorithm of transfer protocols in terms of transfer size, time, and security.

The remainder of this chapter is organized as follows: Section 3.2 presents preliminaries of this work. Section 3.3 discusses the overall architecture of the proposed data minimization algorithm, simulation, and heuristic model's description. Section 3.4 presents the experimental results of the proposed data minimization mechanism including discusses the three transfer protocols behaviors using the standard and the proposed content-encoding schemes with/out two selected compression algorithms: (GZIP [81] and MFCompress [82]) Finally, future work and our conclusions are presented in Section 3.5.

3.2 Preliminaries

Development and implementation of a content-encoding scheme that forms a data-aware protocol derived from HTTP for transferring big genomic datasets, requires an understanding of genomic DNA sets and content-encoding schemes. The next subsections provide a brief description regarding the fundamentals of genomic DNA components and the current use of content-encoding in HTTP.

3.2.1 Genomic DNA

The DNA of any organism is made up of millions of nucleotides monomer building blocks that are joined together to form extremely long polymers. The nucleotides comprise four basic types, abbreviated as follows: A, T, C & G. These nucleotide monomers are joined together in very specific ways to form genes that encode the information required to produce the cells, tissues and organs of an organism. Some of the most important strings of nucleotides are those that together encode the information required to produce proteins, which are the machines of the cells. The average length of the protein coding portion of a gene in humans is 1,100 nucleotides.

In the human genome, it is estimated that there are approximately 20,000 protein coding genes. A rough estimate of the total number of nucleotides of the human genome devoted to encoding for all of the protein machinery required for life is 30 million nucleotides. Abundant additional nucleotides make up the rest of the genome of a human and some of these encode important functional molecules such as various types of RNAs), while the rest are considered non-coding. Altogether, the approximate number of nucleotides making up the 23 chromosomes of the haploid human genome is 3.3 billion nucleotides. Thus, for the entire diploid human genome of 46 chromosomes are 6.6 billion nucleotides. These estimates exclude the mitochondrial genome since it is very small by comparison. In the future, when all human genomes may be sequenced, a large amount of data will be generated and will require efficient management.

Next Generation Sequencing techniques often result in an average coverage of each nucleotide of 30x. Therefore, approximately 180 billion nucleotides of data are obtained every time a human genome is sequenced using existing technologies. If we then scale this estimate up for the entire human population, we project that approximately 1,250 billion nucleotides of data would be generated. By doing simple calculations to estimate how much we need to digitalize human genome (genomic DNA), we derive at the following:

1. Digitalizing a single cell would result in :

   - Single Unit of all DNA (A - C - G - T) - 1 bp.
   - Average Length of exon sequences for one protein-coding gene - (1100 bp).
   - Approximate total length of all exons of protein-coding genes in the human genome (20000 genes - 22000000 bp).
   - Approximate total length of all coding and non-coding DNA in haploid genome (3000000000 bp).
   - Approximate length of diploid human genome. That is all DNA within a single nucleus

(6000000000 bp).

- Approximate number of bp generated in a shotgun sequence using Next Generation Sequencing methods assuming 30x coverage (180000000000 bp).

2. Digitalizing of the world's population genomes (world's population on June 2016 [80]) would result in : Approximate number of bp that would be generated by sequencing diploid genomes for all 7 billion people on earth today (1250000000000 bp).

Hint: 1kilo = 1024 used in these calculations, hence 1 Yotta = 270 KB.

In fact, the growth rate of DNA sequencing over the last 10 years has generated a massive amount of data that doubles approximately every 7 months. According to[9] to date as shown in Table 1.1 on page 10, there are more than 2,500 high-throughput sequencing instruments distributed over 55 countries placed in about 1,000 sequencing centers.

This chapter introduces a new transfer solution for big genomic DNA in a way that maximizes bandwidth utilization and minimizes the transmission size.

3.2.2 Convolutional Neural Network

Convolutional Neural Network (CNN) [83][84] is an example of deep learning (DL) [85] techniques that refer to both deep neural networks and other branches of machine learning, such as deep reinforcement learning. Neural networks are defined as a set of algorithms, modeled loosely after the human brain, and that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or by clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Thus, CNN is defined as an end-to-end system, in which the input is raw data, while the output is a prediction through the distinctive features extracted via intermediate layers. CNN divides into four main layers: convolution, pooling, normalization, and fully connected, as illustrated in Figure 3.2 on page 32 as follows:

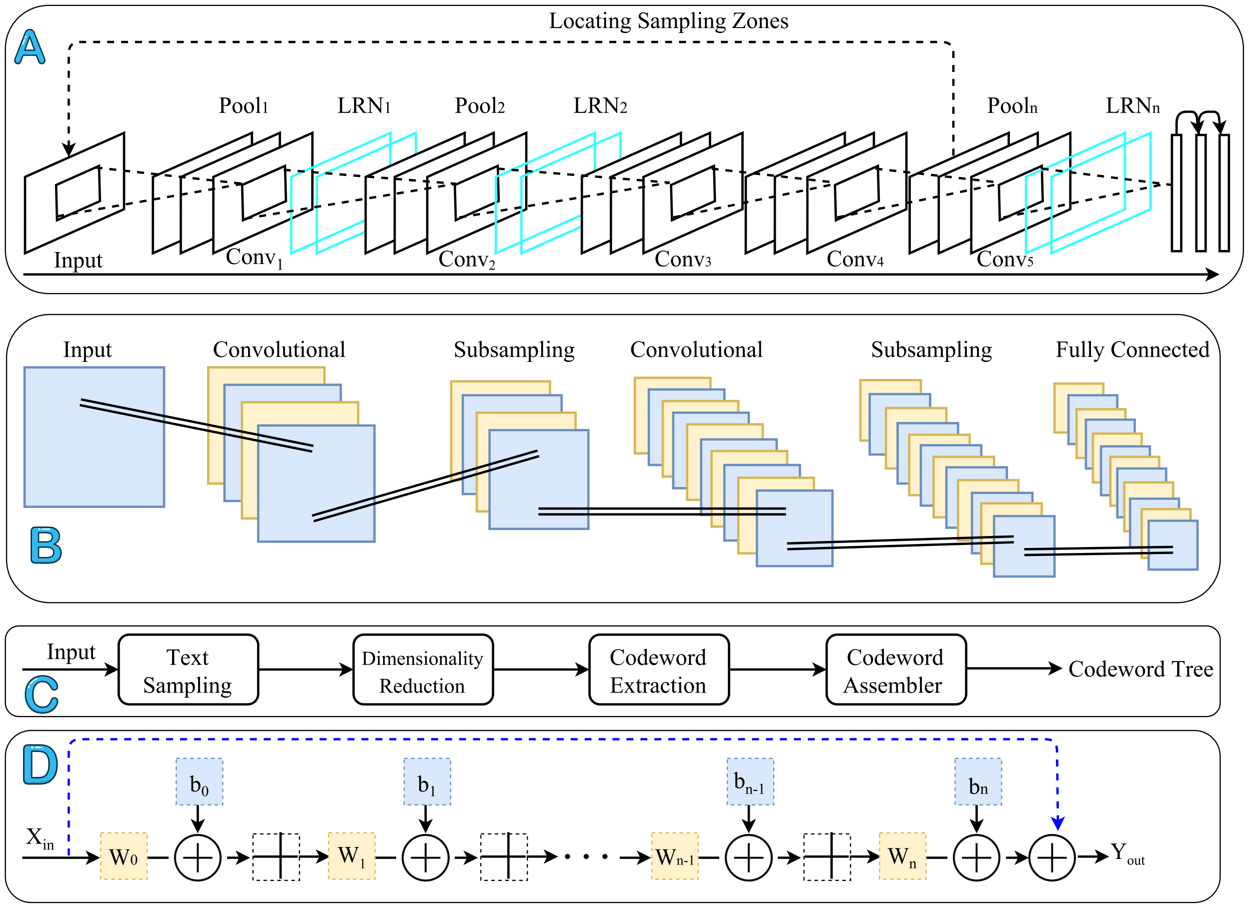


Figure 3.2: Convolutional neural network conceptual model

3.2.2 Convolution Layers

Convolution layers are used to convolve previous layer's feature maps with multiple filter masks to find the most common patterns of tested data. This layer is responsible for running two important jobs: connection and sharing. Connection applies each single convolve filter only to a local region of the input volume and thus, decreases the network weight parameters. Thus, the spatial extent of the local connectivity is sometimes referred to as the receptive field. Sharing the network parameters between layers means it is necessary to use the same filter to convolve the entire feature map at each layer. To formulate the convolution layers, we denote Con_i^l as the i^{th} input feature map of l layer; Ker_{ij}^l refers to connecting the kernels of the feature map of the output layer j^{th} to the feature map of the input layer i^{th} and $bias_j^l$, as an additive bias.; and then the following:

$$Con_j^l = f(\sum_{i=1}^{n} Con_i^{l-1} Ker_{ij}^l + bias_j^l), \tag{3.1}$$

where f denotes the activation function that is usually a rectified linear function.

3.2.2 Pooling Layers

Pooling layers represent the sampling layers that work on combining the outputs of the convolution layers and the related classical spatial pyramid [86]. These layers reduce the spatial size of the feature maps, thus decreasing computation costs in the network. To formulate the pooling layers, we denote as:

$$Con_j^l = next(Con_j^{l-1}), \tag{3.2}$$

where next(Con_j) is a subsampling function of the next layer.

3.2.2 Normalization Layers

Normalization layers are responsible for assembling the output of different layers and provide the best pattern recognition i.e. the most repetition characters. Denoting by a_i the single value of i^{th} feature map, the normalize activity $bias_i$ is given by the expression

$$bias_i = \frac{a_i}{(Ker + \alpha \sum_{j=max(0,1-n/2)}^{min(N-1,i+n/2)} a_j^2)^{\beta}}. \tag{3.3}$$

The constants *Ker*, *n*, α, and β are hyperparameters, and we use *Ker* = 2, *n* = 5, α = 10^{-4}, and β = 0.75 in our experiments.

3.2.2 Fully Connected Layers

The final cycles of normalization produce several fully connected layers that draw the final forms of available data patterns (classifiers). The final layer consists of a combination of the outputs of fully connected layers which generate the new character-codeword tree. The output of character-codeword tree represents the probabilities of the character repetitions, in which the highest one corresponds to the predicted codeword. Then, we can formulate the fully-connected layer,

$$P(y = 1|Con; w) = \frac{1}{1 + \exp(-w^T Con)}, \tag{3.4}$$

where y is label, $x \in R^{(D+1)1}$ represents the D dimensional feature vector, $w \in R^{(D+1)1}$ represents the weight vector, T refers to training data. Considering a classification problem where the response variable y can take any one of N values, we can generalize the binary classification (genomic or non-genomic characters). Thus,

$$P(y = c|Con; W) = \frac{\exp(w_i^T Con)}{\sum_{i=1}^{n} \exp(w_i^T Con)}, \tag{3.5}$$

where $W = [w_1, w_2, ..., w_n] \in R^{(D+1)N}$, each w represents the corresponding category weight parameters.

This subsection presents our implementation of network data minimization solution that relies on the statistical encoding and CNN algorithm via modifying HTTP content-encoding to transfer big genomic datasets expediently and securely. This work assures better performance and bandwidth utilization for the transfer of big genomic datasets via the minimization of data size and time. This model assigns the shortest possible codeword for more symbol occurrences via utilizing a CNN algorithm, as illustrated in section 3.3.3 on page 48, a codeword that divides a dataset into parts and that reads a short string randomly to specify symbol repetitions. Two encoding schemes are used in this model: standard (8-bit) and modified (1,2, and 3-bit). The standard one uses the title (header) of a dataset and other symbols out of the genomic scope i.e. N in the body. The modified encoding scheme uses a convolutional neural network technique for the body of datasets. The proposed encoding scheme starts with initial codewords such those shown in Table 3.3 on page 42 and then periodically updates codewords via a CNN deep learning algorithm, as illustrated in section 3.3.3 on page 48, while using the former codeword table. After each real-time update, the server send the proposed encoding scheme to the client (receiver) prior

Table 3.1: The standard and proposed codewords

Symbol	Frequency	Proposed	Standard
A	0.85	0	01000001
T	0.05	11	01010100
G	0.05	100	01000111
C	0.05	101	01000011
Total bits for 20 symbols	100%	25	160

to applying it on datasets at the server side. Therefore, this model encodes the contents using two main codeword tables: fixed (static) and dynamic, as shown in Figure 4.1 on page 79.

Therefore, encoding 20 symbols in 25 bits yields an average of 1.25 bits/symbol in the proposed example, whereas 160 bits in the current HTTP encoding yields an average of 8 bits/symbol. We designed variable codes in Table 3.3 on page 42 in a way that facilitates decoding, using a prefix property (unambiguously). The prefix property assigns a unique specific bit pattern for each alphabet symbol to ease the decoding operation on the client side. Variable binary encoding is not a new idea; however, it is more common in single or static applications. The use of the proposed encoding scheme requires a prior knowledge to assign short codes for high probabilities; otherwise, short codes would be assigned for rare occurrences (negative results). The current use of HTTP content-encoding is fixed (standard), which means assigning the same weight for each symbol i.e. 8-bits. Therefore, we design a proposed encoding scheme for HTTP in a way that always enables the server to assign short codes for letters that appear more frequently. Although this scheme produces minimum possible bits, it consumes extra time. Our implementation relies on reading parts of the file via the CNN algorithm to estimate symbol frequencies and to set codes. Consequently, we can get minimum possible codes in less time via applying the CNN algorithm to the proposed scheme. Time complexity is *O(n)*. This encoding approach works well with high symbol repetition occurrences, so that assigning a 1-bit length codeword for the highest occurrence symbol, a 2-bit length codeword for less repetition, and a 3-bit codeword length for the remaining 2 symbols is an optimal approach. The pseudocode for our protocol is highlighted in algorithm 4

Algorithm 1 Dynamic variable-length binary encoding

```
1: procedure ENCODING
2:     DVLBE.doSamplingAndBuildFreqArray(inputStream , fileParti-tions,SamplingRatioPerPartition)
3:     if inputStream.hasGenomeFileheader then
4:         outputStream.write(GenomeFileheader)
5:     end if
6:     DVLBE.writeGenomeSymbolsEncoder(outputStream)
7:     while !inputStream.EOF do
8:         genomeChar $\leftarrow$ inputStream.GetChar().
9:         code $\leftarrow$ DVLBE.encode(genomeChar).
10:        oneByteStore.store(code).
11:        if oneByteStore.ISFull() then
12:            outputStream.write(oneByteStore).
13:            oneByteStore.empty().
14:        end if
15:    end while
16:    if !oneByteStore.ISEmpty() then
17:        outputStream.write(oneByteStore).
18:        outputStream.write(NumOfExtraBits).
19:    end if
20: end procedure
1: function DOSAMPLINGANDBUILDFREQARRAY(INPUTSTREAM , FILEPARTI-TIONS,SAMPLINGRATIOPERPARTITION)
2:     PartitionSize $\leftarrow$ FileSize / filePartitions().
3:     SamplesPerPartition $\leftarrow$ PartitionSize * SamplingRatioPerPartition.
4:     SymbolsFrequencyArray $\leftarrow$ Interger Array with Four elements filled with zeros.
5:     while filePartitions >0 do
6:         Offest $\leftarrow$ Random.GetDouble * PartitionSize .    ▷ Random.GetDouble generates random numbers between 0 and 1
7:         inputStream.Seek(offest).
8:         inputStream.Read(SamplesPerPartition, dataBuffer).
9:         UpdateFrequency(SymbolsFrequencyArray, $\leftarrow$ dataBuffer).
10:        filePartitions $\leftarrow$ filePartitions - 1.
11:    end while
12:    FreqArray.Build(SymbolsFrequencyArray). return FreqTable
13: end function
```

on page 81. In addition, we implemented a variable-length binary encoding (arbitrary or bind) to compare with our proposed and standard.

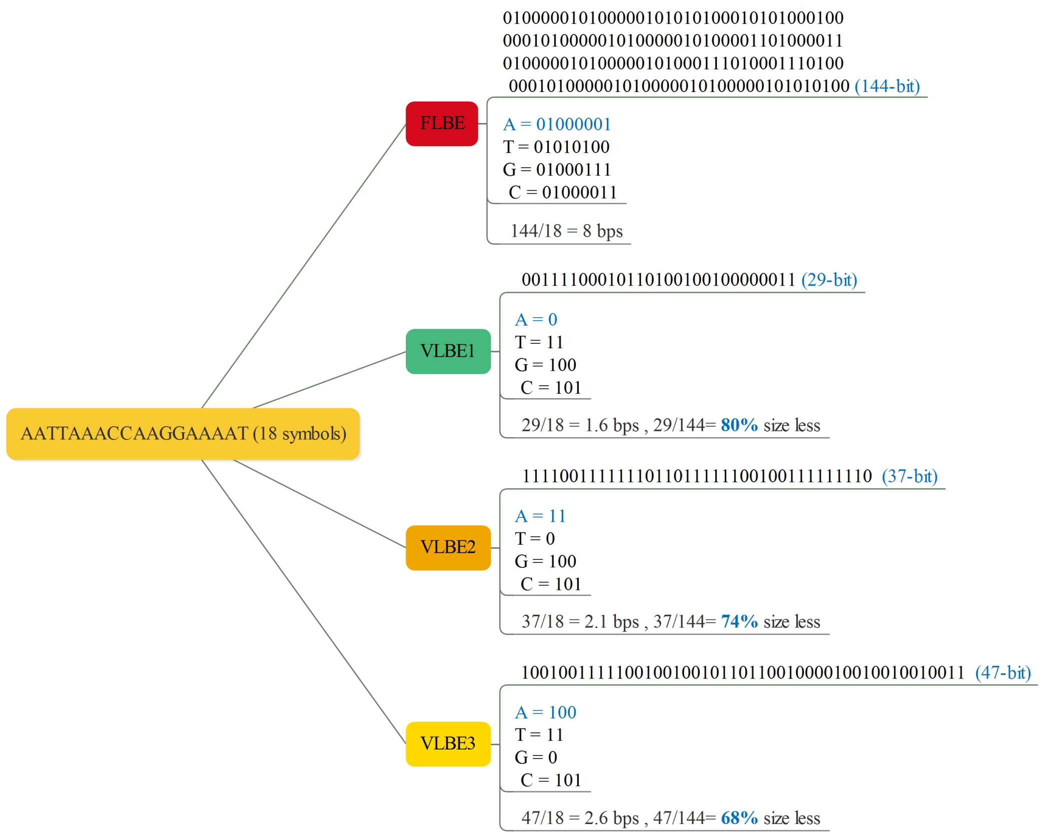
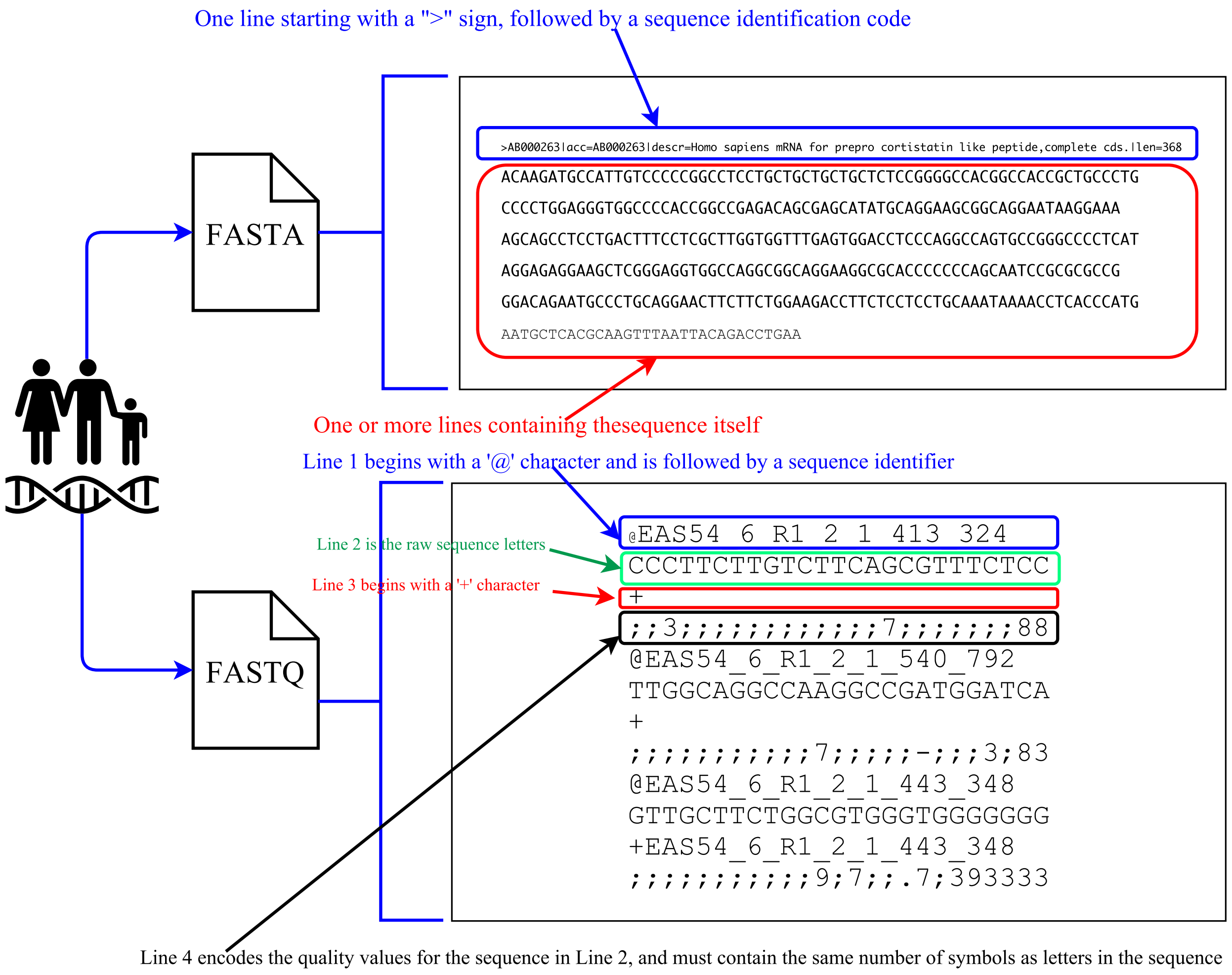

AATTAAACCAAGGAAAAT (18 symbols)
FLBE
010000010100000101010100010101000100
000101000001010000010100001101000011
010000010100000101000111010001110100
000101000001010000010100000101010100 (144-bit)
A = 01000001
T = 01010100
G = 01000111
C = 01000011
144/18 = 8 bps
VLBE1
00111100010110100100100000011 (29-bit)
A = 0
T = 11
G = 100
C = 101
29/18 = 1.6 bps , 29/144= 80% size less
VLBE2
1111001111111011011111100100111111110 (37-bit)
A = 11
T = 0
G = 100
C = 101
37/18 = 2.1 bps , 37/144= 74% size less
VLBE3
10010011111001001001011011001000010010010010011 (47-bit)
A = 100
T = 11
G = 0
C = 101
47/18 = 2.6 bps , 47/144= 68% size less

Figure 3.3: Example of the HTTP content encoding schemes for a string of 20 symbols with variety of repetitions. 3 possible codes for the genomic symbols [A, C, G and T]

3.2.3 Genomic DNA Generator

In this section, we discuss a genomic *DNA* generator we implemented to generate *FASTA* format genomic files [87]. FASTA file is a single sequence described by a title line followed by one or more data lines. The title line begins with a right-angle bracket followed by a label. The label ends with the first white space character. Everything after that on the first line is considered a comment. The data lines begin right after the title line and contain the sequence characters in order, as shown in Figure 3.4 on page 38.



One line starting with a ">" sign, followed by a sequence identification code
FASTA
One or more lines containing thesequence itself
Line 1 begins with a '@' character and is followed by a sequence identifier
Line 2 is the raw sequence letters
Line 3 begins with a '+' character
FASTQ
Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence

```
>AB000263|acc=AB000263|descr=Homo sapiens mRNA for prepro cortistatin like peptide,complete cds.|len=368
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTG
CCCCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAA
AGCAGCCTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCAT
AGGAGAGGAAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCG
GGACAGAATGCCCTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATG
AATGCTCACGCAAGTTTAATTACAGACCTGAA
```

```
@EAS54 6 R1 2 1 413 324
CCCTTCTTGTCTTCAGCGTTTCTCC
+
;;3;;;;;;;;;;;;7;;;;;;;88
@EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
+
;;;;;;;;;;;7;;;;;-;;;3;83
@EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
+EAS54_6_R1_2_1_443_348
;;;;;;;;;;;9;7;;.7;393333
```

Figure 3.4: FASTA file format that includes living organism's genomic datasets. Each data line except the last should be exactly 60 letters long, although many programs allow some flexibility on that score

We generated 18 FASTA files in different sizes and symbol frequencies to assess the results of our proposed HTTP content-encoding, which works efficiently for more occurrence symbols as shown in Table 4.2 on page 87. The genomic generator (GG) runs in two modes: *auto* and *manually* as can be seen in Algorithm 3 on page 68.

In the auto mode, the GG takes one input parameter that is a needed file size and produces a FASTA file with almost equal symbol repetitions i.e. 25% each. In the manual mode, the GG takes three input parameters: needed *file size*, *symbol* and needed *repetition* for this symbol

Table 3.2: Generated FASTA files using a genomic generator

Datasets	Number of DNA Symbols in each Dataset							
	A	C	G	T	A%	C%	G%	T%
100M[25]	26,843,545	26,843,645	31,808,042	21,878,898	0.25%	0.25%	0.30%	0.20%
100M[35]	37580963	30868915	13425306	25498,926	0.35%	0.28%	0.13%	0.24%
100M[50]	53,687,091	14,786,579	25,675,457	13,224,952	0.50%	0.14%	0.24%	0.12%
500M[25]	134,217,728	149,129,757	149,112,975	104,410,196	0.25%	0.28%	0.28%	0.19%
500M[35]	187,904,819	153,550,318	116,324,344	79,091,072	0.35%	0.29%	0.21%	0.15%
500M[50]	268,435,456	93,792,797	100,263,349	74,378,798	0.50%	0.17%	0.19%	0.14%
1G[25]	268,435,456	219,637,924	292,809,252	292,858,680	0.25%	0.21%	0.27%	0.27%
1G[35]	375,809,638	221,793,199	254,370,797	221,767,473	0.35%	0.20%	0.24%	0.21%
1G[50]	536,870,912	203,878,326	101,922,932	231,068,630	0.50%	0.19%	0.09%	0.22%
5G[25]	1,342,177,280	1,283,854,849	1,517,198,501	1,225,475,930	0.25%	0.24%	0.28%	0.23%
5G[35]	1,879,048,192	1,143,192,988	1,203,320,398	1,143,143,958	0.35%	0.21%	0.22%	0.22%
5G[50]	2,684,354,560	723,874,757	1,176,284,699	784,189,984	0.50%	0.13%	0.22%	0.15%
10G[25]	2,684,354,560	2,467,876,125	2,338,051,024	3,247,131,411	0.25%	0.23%	0.22%	0.30%
10G[35]	3,758,096,384	2,594,087,000	2,532,325,440	1,852,902,248	0.35%	0.24%	0.24%	0.17%
10G[50]	5,368,709,120	894,831,061	2,013,252,324	2,460,615,495	0.50%	0.08%	0.19%	0.23%
50G[25]	13,421,772,800	13,421,823,505	14,380,428,192	12,463,041,103	0.25%	0.25%	0.27%	0.23%
50G[35]	18,790,481,920	10,550,108,277	9,738,567,871	14,607,897,292	0.35%	0.20%	0.18%	0.27%
50G[50]	26,843,545,600	10,949,303,940	7,064,179,103	8,830,011,357	0.50%	0.20%	0.14%	0.16%

and generates a FASTA file with required repetitions. The manual GG controls 1 symbol repetition only, and the rest generate randomly at different rates. To simplify our experiments, we generate 18 different FASTA files with different frequency rates and sizes up to 50-gigabyte. For example, we generated a 10GB FASTA file in 3 different frequency rates: 25%, 35% and 50% for the symbol (A). Symbol repetition aims to verify our implementation improvement in contrast to the HTTP encoding for big genomic datasets from size and time perspectives. The first scenario assumes 25% occurrence rate for each symbol in a dataset. The second scenario assumes a 35% repetition rate for one symbol of a dataset and 65% occurrences for the 3 other symbols. The third scenario assumes a 50% repetition for one symbol, while the 3 other symbols have different repetitions within 50% of the dataset. The goals of the GG implementation are to customize file sizes and occurrence rates. Also, the GG aims to avoid privacy violations to validate our content-encoding method, in contrast to other possible approaches during big genomic transferring. The GG algorithm can be seen in Algorithm 3 on page 68. Note: A file of 25% of symbol (A) does not necessarily mean the rest of the symbols have the same symbol repetitions i.e. 25% for (C), 25% for (G), and 25% for (T), it may vary.

3.3 A Dynamic Model of Content-Encoding

In this section, we illustrate our three implementation versions of the content-encoding: FLBE, fixed VLBE (FVLBE), and dynamic VLBE (DVLBE). Also, model formulations are discussed in this section for different possible scenarios of symbol repetitions to verify how our proposed encoding scheme compares to the current transfer protocols i.e. HTTP and FTP content-encoding scheme. The encoding model description and formulation are presented in the following subsections:

3.3.1 Model Description

This subsection presents our implementation of an adaptive data-aware transfer protocol that modifies HTTP content-encoding to transfer big genomic DNA datasets. This work has been done to assure better performance and bandwidth utilization for big data transfer via minimizing data size and time. This model assigns the shortest possible codeword for more symbol occurrences via utilizing a machine learning random sample, as illustrated in section 3.3.3 on page 48, a codeword that divides a dataset into parts and that reads a short string randomly to specify symbol repetitions.

Two encoding schemes are used in this model: standard (fixed) and modified (dynamic). The standard one uses the title (header) of a dataset and other symbols out of the genomic scope i.e. N in the body. The modified encoding scheme used a convolutional neural network technique for the body of datasets. It starts with initial codewords such those shown in Table 3.3 on page 42 and then periodically updates codewords via random sample of machine learning, as illustrated in section 3.3.3 on page 48, while using the former codeword table. After each real-time update, it is sent first to the client (receiver) prior to applying it on datasets at the server side.

Therefore, this model encodes the contents using two main codeword tables: fixed and dynamic (changes many times), as shown in Figures 3.6 on page 49, 3.7 on page 50, and 3.8 on page 51.

The fact that the genomic DNA alphabet consists of only four symbols, inspired us to build an adaptive encoding scheme to speed up a transfer time. This implementation combines a VLBE scheme and dynamic behavior to guarantee producing the best updatable VLBE scheme for any genomic file all the time. Producing an updatable VLBE scheme assures getting the minimum possible data that minimizes transfer time. The genomic alphabet (A) is comprised of four symbols: $\{$A, T, G, C$\}$, which can be presented in less than an 8-bit codewords length as used in the current implementation. We can encode the genomic dataset symbols in four unique decipherable codewords i.e. [0, 11, 100, 101] or simply [0, 3, 4, 5] as shown in Table 3.3 on page 42. Traditional HTTP works in *12 or 14 (if compression is used) steps*, but our implementation of HTTP using the proposed encoding scheme works in *13 steps* via adding a sampling sub-step to shorten the transfer time.

Step 7 divides into 2 sub-steps: the first sub-step achieves the sampling and the assignment of codewords in the second sub-step. The first sub-step requires a brief period of time at the beginning, but reduces a significant amount of time for the next steps. The HTTP in most browsers starts when clients search for specific data. The HTTP initiates a connection with the server that contains the required data. The connection between the client and the server establishes what is called a 3-way handshake through the TCP/IP protocol.

After establishing the connection, the client sends a request for certain resource(s) to the server that checks the header(s), the method(s), and the resource address(es). The server retrieves the required data and starts to convert file symbols to the binary form, using a FLBE and passes it to a compression (optional) i.e. GZIP or MFCompress algorithms. At this point, our main contribution takes place by doing a random sampling using a machine learning algorithm as illustrated in section 3.3.3 on page 48 for each dataset.

The result of the sampling phase is stored in an array to calculate symbol repetitions and create the encoding tree that is used in the content-encoding phase. After building the encoding tree, the server starts to encode the contents and sends the data (response) through network medium i.e. Ethernet along with this tree that will be used in the decoding phase. The client receives the dataset from the server, including the communication header(s), encoding tree, and method(s) to verify and authenticate the source, and then continues the communication.

We utilize a binary tree as a structure to represent our DVLBE because it is faster to search, avoids duplicate values, and facilitates decoding at the receiver side. Also, the use of a binary tree as a structure gives the programmer special flexibility because it offers one of the two paths to follow and cuts the search time to half, thereby increasing process throughput.

A simple example is listed below to theoretically assess our binary encoding in contrast to current use of HTTP binary encoding for 20 genomic symbols string based on Table 3.3 on page 42.

Table 3.3: The fixed and variable codewords

Symbol	Frequency	FVLBE	HTTP
A	0.49	0	01000001
T	0.25	11	01010100
G	0.25	100	01000111
C	0.01	101	01000011
Total bits for 20 symbols	100%	37	160

Therefore, encoding 20 symbols in 37 bits yields an average of 1.85 bits/symbol in FVLBE, whereas 160 bits in current HTTP encoding yields an average of 8 bits/symbol. However, it is not easy to assign the shortest code for more frequently repeated letters using FVLBE for real-time applications. We designed variable codes in Table 3.3 on page 42 in a way that makes it easy to decode using a prefix property (unambiguously). The prefix property assigns a unique specific bit pattern for each alphabet symbol to ease the decoding operation in the client side. Variable binary encoding is not a new idea; however, it is more common in single or static applications. FVLBE needs a prior knowledge to assign short codes for high probabilities; otherwise, short

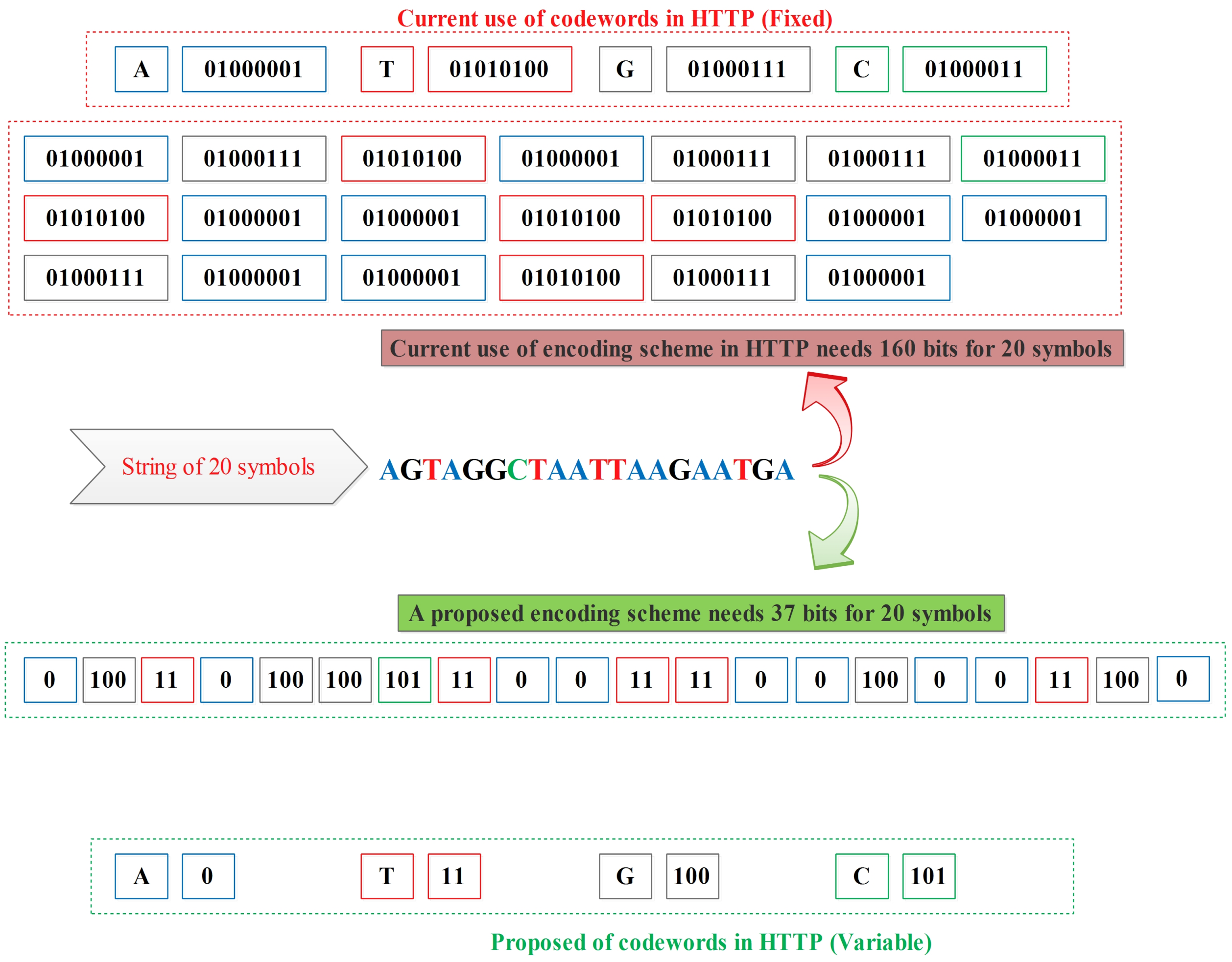


Figure 3.5: Example of the HTTP binary encoding schemes for a string of 20 symbols with variety of repetitions. The top encoding scheme represents current use in HTTP (fixed), while the bottom scheme presents a fixed of variable-length binary encoding in a range of (1-3) bits

codes would be assigned for rare occurrences (negative results).

Implementing a data transfer protocol with FVLBE will not guarantee producing the minimum possible codewords since files do not have the same frequencies. Current use of HTTP content-encoding is fixed (FLBE), which means assigning the same weight for each symbol i.e. 8-bits. Therefore, we design a DVLBE for HTTP in a way that always enables the server to assign short codes for letters that appear more frequently. DVLBE performs a two-cycle work to assign the shortest code: during the first cycle, it reads the whole file to build a codewords table and sets the best prefix codes for that file. In the second cycle, it substitutes codewords for each symbol based on the codewords table. Although this scheme produces minimum possible bits, it consumes extra time.

Our implementation relies on reading parts of the file via random sampling to estimate symbol frequencies and to set codes. Consequently, we can get minimum possible codes in less time via applying sampling to the DVLBE. Time complexity is *O(n)*. This encoding approach works perfectly with high symbol repetition occurrences, so that assigning a 1-bit length codeword for the highest occurrence symbol, a 2-bit length codeword for less repetition, and a 3-bit codeword length for the remaining 2 symbols is an ideal approach. The pseudocode for our protocol is highlighted in Algorithm 4 on page 81. In addition, we implemented a variable-length binary encoding (arbitrary or bind) to compare with our DVLBE and original HTTP FLBE.

3.3.2 Problem Formulation

Our analysis starts from the fact that in practical cases, the transfer time of data fluctuates due to several reasons, such as bandwidth and message loss. The data transfer throughput thr measured by the minimum needed time (*t*) to transfer certain data amount (N) from the sender to the receiver. In order to minimize transfer time, we need to either maximize the bandwidth or minimize data volume during transferring. This can be formalized in the following Equations: 6.6 and 3.7 on page 119:

$$thr = \frac{N}{t} \tag{3.6}$$

A higher thr means better protocol throughput via transferring large data amounts in a time unit. The protocol throughput is inversely proportional to the number of bits per symbol (nucleotide). For example, transferring N string symbols in B bits indicates the efficiency of the encoding scheme as shown in the following:

$$bpn = \frac{B}{N} \tag{3.7}$$

Algorithm 2 Dynamic variable-length binary encoding

```
1: procedure ENCODING
2:     DVLBE.doSamplingAndBuildFreqArray(inputStream , fileParti-
   tions,SamplingRatioPerPartition)
3:     if inputStream.hasGenomeFileheader then
4:         outputStream.write(GenomeFileheader)
5:     end if
6:     DVLBE.writeGenomeSymbolsEncoder(outputStream)
7:     while !inputStream.EOF do
8:         genomeChar ← inputStream.GetChar().
9:         code ← DVLBE.encode(genomeChar).
10:        oneByteStore.store(code).
11:        if oneByteStore.ISFull() then
12:            outputStream.write(oneByteStore).
13:            oneByteStore.empty().
14:        end if
15:    end while
16:    if !oneByteStore.ISEmpty() then
17:        outputStream.write(oneByteStore).
18:        outputStream.write(NumOfExtraBits).
19:    end if
20: end procedure
1: function DOSAMPLINGANDBUILDFREQARRAY(INPUTSTREAM , FILEPARTI-
   TIONS,SAMPLINGRATIOPERPARTITION)
2:     PartitionSize ← FileSize / filePartitions().
3:     SamplesPerPartition ← PartitionSize * SamplingRatioPerPartition.
4:     SymbolsFrequencyArray ← Interger Array with Four elements filled with zeros.
5:     while filePartitions >0 do
6:         Offest ← Random.GetDouble * PartitionSize .    ▷ Random.GetDouble generates
   random numbers between 0 and 1
7:         inputStream.Seek(offest).
8:         inputStream.Read(SamplesPerPartition, dataBuffer).
9:         UpdateFrequency(SymbolsFrequencyArray, ← dataBuffer).
10:        filePartitions ← filePartitions - 1.
11:    end while
12:    FreqArray.Build(SymbolsFrequencyArray). return FreqTable
13: end function
```

Here, the minimum *bpn* is better and positively reflects on the transfer time. For instance, if the transfer time for 1 bit is (T) time unit, then the transfer time for 1 nucleotide that consists of 8 bits equals to $8T$ time units. Therefore, minimizing the *bpn* reduces the data amount which reduces required transfer time. In the example mentioned in section 4.2.1 on page 78, we calculated 1.85.

Consequently, 1.8 T is a significant reduction (6.15T) in contrast to the current HTTP encoding (8T).

In this encoding implementation, we divide the file of N strings into S slots to apply sampling on each slot and store the output in a frequency array. After that, we calculate the symbol repetitions to build the encoding tree, as shown in the equations 3.8 and 3.9 on page 46 and Figures 3.6 on page 49, 3.7 on page 50, and 3.8 on page 51:

$$P_{file} = \frac{N}{s} \tag{3.8}$$

$$Tree_{encoding} = \frac{Nr}{s} \tag{3.9}$$

Where $Tree_{encoding}$ refers to the binary encoding tree, the P_{file} represents file parts or slots. The *r* refers to a sampling ratio that applies to each file slot or part e.g. 1%.
By convention, the left child is labeled 0 and the right child is labeled 1 as shown in Figures 3.6 on page 49, 3.7 on page 50, and 3.8 on page 51.

The DVLBE scheme adds an extra phase (sampling) that requires more time to use the machine learning algorithm as illustrated in section 3.3.3 on page 48 and that builds the encoding tree. However, the sampling phase has a significant impact on the next phases (binary encoding, compression, transfer, decompression, decoding to plain text). Therefore, the time penalty of the sampling phase can be remedied by significant time reduction in the overall transfer time via minimizing a binary form of datasets, which is the contribution of this work.

To simplify our model, let us assume we have an (A) alphabet as $\{a_1, a_2, a_3, ..., a_n\}$ that consists of *n* symbols, for the genomic dataset, A $\in$ {A, T, G, C}(nucleotides). Codeword presented in C(A) = {0,11,100,101}. The *time* complexity for encoding a string of N symbols using the HTTP (fixed) encoding is O(N), and the *space* complexity S can be calculated in the equation 3.10 on page 47:

$$S_{flbe} = \sum_{i=1}^{n} 8a_i \tag{3.10}$$

The space complexity of our proposed encoding scheme in the three possible cases (worst, average and best) can be formulated in equations 3.11, 3.12 and 3.13 on page 47:

- Worst case would include an enhancement rate of 0.625

$$S_{worst}(A) = \sum_{i=1}^{n} 3a_i \tag{3.11}$$

- Average case would include an enhancement rate of 0.71875

$$S_{average}(A) = \sum_{i=1}^{n} 2.25a_i \tag{3.12}$$

- Best case would include an enhancement rate of 0.875

$$S_{best}(A) = \sum_{i=1}^{n} a_i \tag{3.13}$$

The 3 mentioned scenarios have a time complexity of O(N/P), where the P $\in \{1, 3\}$. Also, we can divide and formulate our model costs (Co) in equations 3.14, 3.15, 3.16, 3.17, and 3.18 on page 48:

$$Co_{total} = Co_{computation} + Co_{communication} \tag{3.14}$$

$$Co_{computation} = O\Bigg(Co_{header\ check} + Co_{encoding}\Bigg) \tag{3.15}$$

$$Co_{encoding} = Co_{sampling} + Co_{converting} \tag{3.16}$$

$$Co_{sampling} = Co_{data\ picking} + Co_{codeword\ assigning} \tag{3.17}$$

$$Co_{communication} = O\left(Co_{3\ way\ handshake} + Co_{bandwidth}\right) \tag{3.18}$$

The previous equations depict the total cost of our encoding implementation for data transfer over HTTP. As mentioned above, our model adds extra time for sampling to build the encoding tree, but significantly reduces the overall transfer time, which reaches to 99 times faster compared to the current encoding scheme. The client (browser) requests a genomic dataset from the server after establishing a connection via 3-way handshaking. The server checks header(s), locates the file address, and then assigns codewords for each nucleotide. After that, it encodes data onto a binary form to transfer it over the medium. Also, the proposed encoding scheme uses machine learning algorithm, as illustrated in section 3.3.3 on page 48 to assign less codewords for more frequent symbols, as illustrated in the next section.

3.3.3 Heuristic Model

One of our innovations is that we can automatically locate the important regions which contribute more for recognition, i.e., more discriminative among these regions. Because the last convolution layer extracts more discriminative and semantic features, we use feature maps of the last convolution layer to locate interesting parts, and this procedure is marked with dash line in Figure 3.2 on page 32. There are multiple feature maps with 64 x 64 size and we simply calculate the average value of these maps to achieve a new feature map, i.e., each sample value in this new feature map is the mean of the corresponding sample in the several feature maps. As demonstrated in Figure 3.2 on page 32, we use this process to all training codeword arrays to obtain the average feature map of the entire training set.

In this chapter, a novel encoding scheme is proposed that significantly reduces data transfer as can be seen in Figures 3.6 on page 49, 3.7 on page 50, and 3.8 on page 51. Two

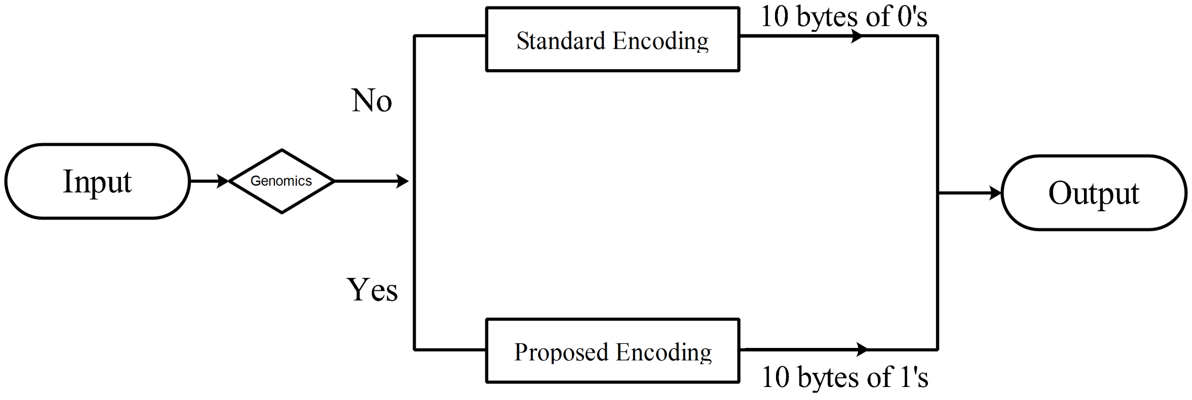


Figure 3.6: A proposed data minimization scheme during transfer stage

encoding sets are used in this scheme: standard and alternative (proposed) set. The alternative symbol encodings are created via an encoding generator $\mathscr{E}$ that runs a series of random samplings during the data transfer process. The encoding generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$ consists of two main processes: random sampling $\mathscr{E}$ and encoding improvement function f_i, where $\mathscr{E}_l$ is the last symbol in the encoding codewords. The input symbols m, the last generated array of symbol codewords a_l, current array of symbol codewords a_c and output bits n.

The following two theorems illustrate the proposed encoding scheme. The encoding generator discussed in theorem 3.1 and sampling improvement function discussed in theorem 3.2.

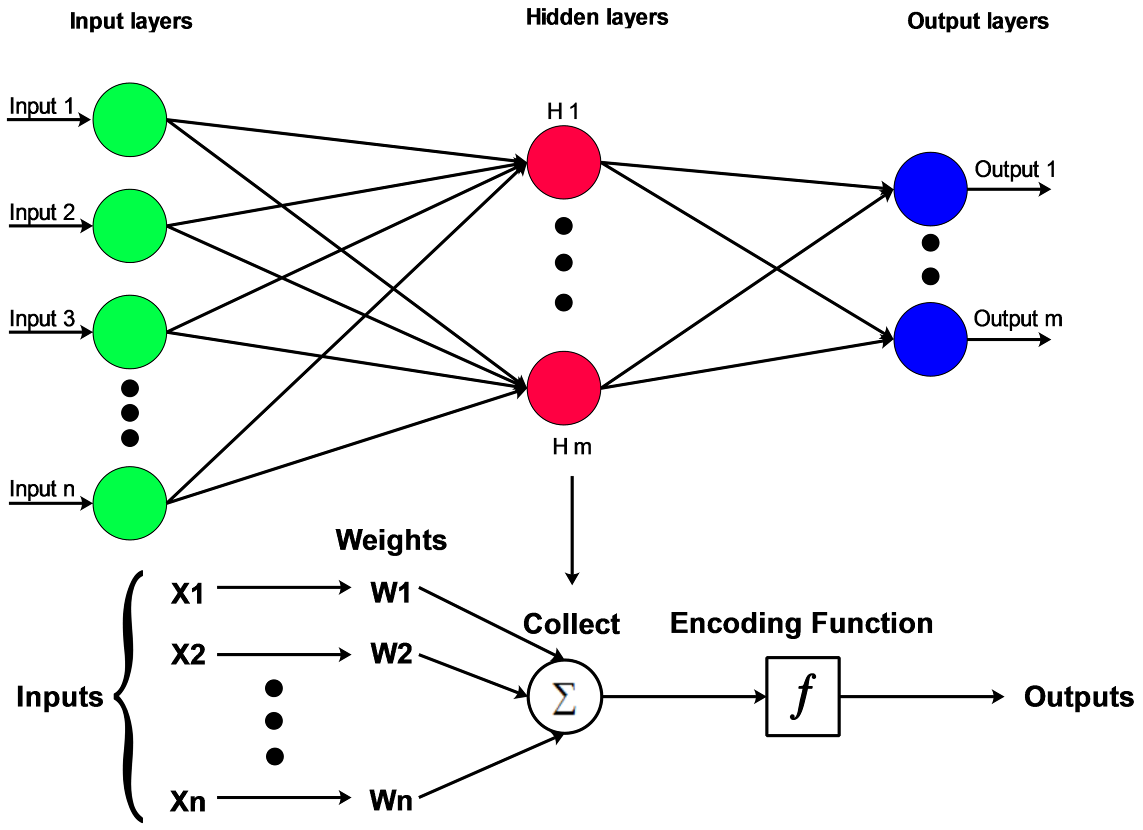


Figure 3.7: The internal process of the proposed data minimization scheme

Theorem 3.1

The alternative symbols encoding E_{S_n} generates minimum possible variable-length code-

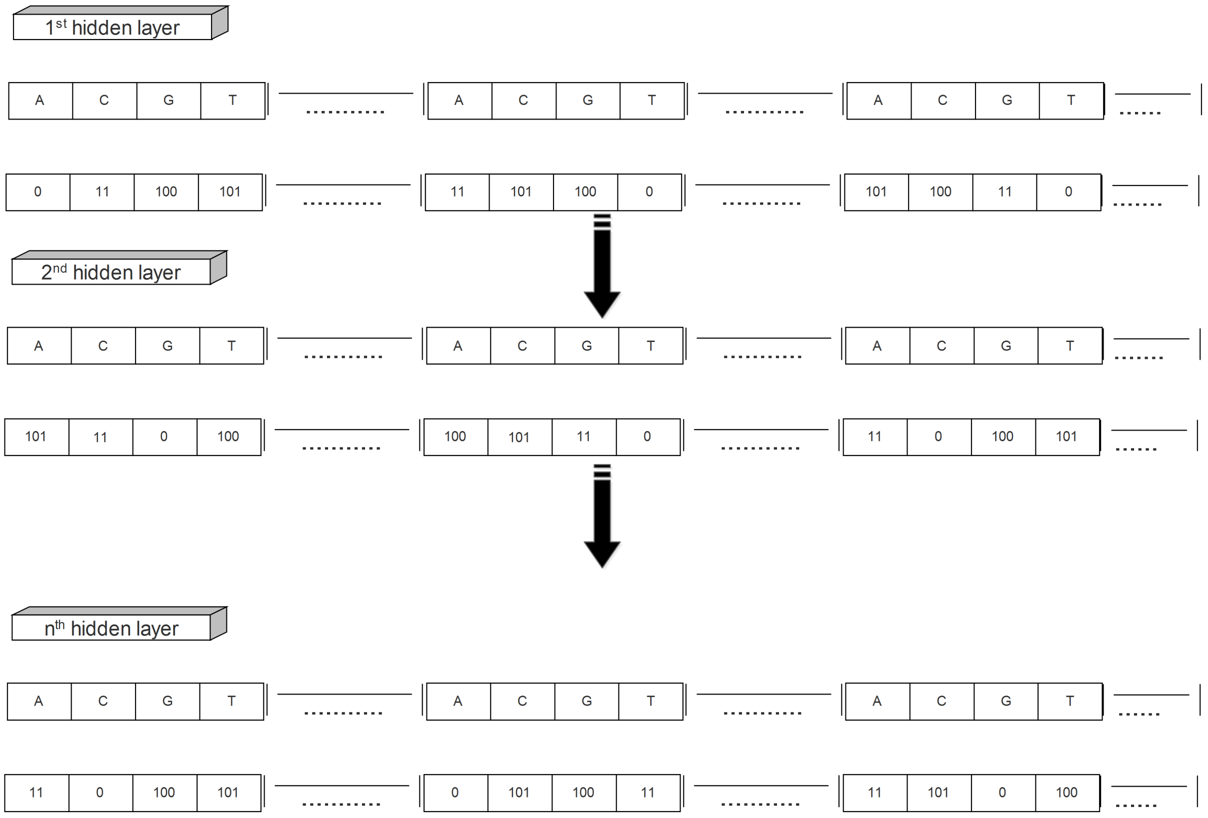


Figure 3.8: A codewords generator architecture

words for alphabet symbols via running series of random sampling S_n over encoding generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$. For $S_n \geq 1$.

we have

$$E_{S_n} = \begin{pmatrix} n \\ S_n \end{pmatrix} 2^{-r(1-a_l)} \sum_{i=1}^{r} \begin{pmatrix} r \\ i \end{pmatrix} p_i^{S_n} (1-p_i)^{n-S_n} \tag{3.19}$$

where

$$p_k = \sum_{j=1}^{S_c} f_{ij} \sum_{i=1}^{B_c} \frac{\begin{pmatrix} j \\ i \end{pmatrix} \begin{pmatrix} r-j \\ k-i \end{pmatrix}}{\begin{pmatrix} r \\ k \end{pmatrix}}$$

For series random sampling in the encoding generator, we have

$$E_{S_n} = \sum_{i=1}^{S_n} \frac{E_i^l E_i^c}{\begin{pmatrix} S_n \\ i \end{pmatrix}} \tag{3.20}$$

where E_i^l is the average of output symbol in last sampling cycle, and E_i^c is the average of current sampling cycle. We can use [88] to formulate the E_i^l as

$$E_i^l = \begin{pmatrix} r \\ i \end{pmatrix} 2^{-r(1-a_l)} \tag{3.21}$$

We denote by r for the sampling size, p_{C_r,C_b} for the probability of *n* input symbols (C_n) have character repetitions (C_r) with character bits (C_b), i.e.,

$$p_r(C_i) := Pr\{C_i = x | Character_{repetition}(C_i)$$

$$= C_r, Character_{bits}(C_i) = C_b\}$$

for any *i* ∈ {1, . . . , *n*} and *x* ∈ { A, C, T, G}. We may write this probability as

$$p_{C_r,C_b}(C_n) = \sum_{i=1}^{n} \frac{\begin{pmatrix} C_r \\ i \end{pmatrix} \begin{pmatrix} r - C_r \\ C_b - i \end{pmatrix}}{\begin{pmatrix} r \\ C_b \end{pmatrix}}$$

For simplicity, we ignore C_r consideration to obtain

$$p_{C_b}(C_n) := Pr\{C_i = x | Character_{bits}(C_i) = C_b\}$$

for any *i* ∈ {1, . . . , *n*}. If we have an ideal sampling size S_n is equal to alphabet symbols and distributed binomially, then we can easily find codewords of each as

$$p_{C_b}(C_{S_n}) := Pr\{C_i = x | C_b(C_i) = C_b\}$$

$$= \begin{pmatrix} n \\ S_n \end{pmatrix} p_{C_b}^{S_n}(1 - p_{C_b})^{n-S_n} \tag{3.22}$$

also we may write it as

$$p_{C_b}(C_{S_n}) = \sum_{j=1}^{S_n} f_{ij} p_{j,C_b}(C_j) \tag{3.23}$$

Then we can find the average of symbol repetitions of current sampling cycle by multiplying (3.22) by C_b, to get

$$E_i^c = \begin{pmatrix} r \\ C_b \end{pmatrix} \begin{pmatrix} n \\ S_n \end{pmatrix} p_{C_b}^{S_n}(1 - p_{C_b})^{n-S_n} \tag{3.24}$$

Therefore, we proved (3.19) by using (3.20), (3.21) and (3.24).

Next, we discuss the sampling improvement function f_i of $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$. To clarify our theorem, we denote by α for the current sampling improvement and β for the last sampling improvement. The comparison function of encoding generator can be written by

$$f_i(\alpha) = \lim_{n\to\infty} \frac{1}{\boldsymbol{n}} \log_2 E_{\alpha n} \tag{3.25}$$

Theorem 3.2

The sampling improvement function f_i for symbols that are randomly picked by the encoding generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$ is always assigns minimum possible codewords for symbols as can be formatted by

$$f_i(\alpha) = C_b(\alpha) - a_c(1-a_l) + f_{min}(\alpha) \tag{3.26}$$

$f_{min}(\alpha)$ can be written as

$$f_{min}(\alpha) := \min f(\alpha, \beta),$$

We denote by β for the symbol codewords of last sampling cycle. We also define $f(\alpha, \beta)$ by

$$f(\alpha, \beta) := a_c C_b(\beta) + \alpha \log_2 \gamma + (1-\alpha)\log_2(1-\gamma),$$

We can define the sampling improvement factor γ by

$$\gamma := \frac{1}{2}\sum_{j=1}^{n} f_{ij}\left[1-(1-2\beta)^j\right]$$

for any $j \in \{1, \ldots, n\}$

Let assume we have input symbols i.e., $\mathbb{N}_r^*$ = { 1, 2,..., r} and from (3.19) and (3.25) we have

$$\begin{aligned}
&= \frac{1}{n}\log_2 E_{\alpha n} \\
&= \frac{1}{n}\log_2 \begin{pmatrix} n \\ \alpha n \end{pmatrix} - a_c(1-a_l) \\
&+ \frac{1}{n}\log_2 \sum_{i=1}^{r} \begin{pmatrix} r \\ i \end{pmatrix} p_i^{S_n}(1-p_i)^{n-S_n} \qquad (3.27) \\
&= C_b(\alpha) - \frac{1}{2n}\log_2(2\pi n\alpha(1-\alpha)) - a_c(1-a_l)
\end{aligned}$$

$$+\frac{1}{n}\log_2\sum_{i=1}^{r}\begin{pmatrix} r \\ i \end{pmatrix} p_i^{S_n}(1-p_i)^{n-S_n} \tag{3.28}$$

$$= C_b(\alpha) - \frac{1}{2n}\log_2(2\pi n\alpha(1-\alpha)) - a_c(1-a_l)$$

$$+\frac{1}{n}\log_2(a_c n) + \frac{1}{n}\log_2\begin{pmatrix} r \\ i \end{pmatrix} p_i^{S_n}(1-p_i)^{n-S_n} \tag{3.29}$$

$$= C_b(\alpha) - \frac{1}{2n}\log_2(2\pi n\alpha(1-\alpha)) - a_c(1-a_l) + \frac{1}{n}\log_2(a_c n)$$

$$+a_c C_b\begin{pmatrix} i \\ r \end{pmatrix} - \frac{1}{2n}\log_2\left(2\pi a_c n\frac{i}{r}\left(1-\frac{i}{r}\right)\right)$$

$$+\alpha\log_2 p_i + (1-\alpha)\log_2(1-p_i)$$

$$= C_b(\alpha) - \frac{1}{2n}\log_2(2\pi n\alpha(1-\alpha)) - a_c(1-a_l) + \frac{1}{n}\log_2(a_c n)$$

$$+a_c C_b(\beta) - \frac{1}{2n}\log_2(2\pi a_c n\beta(1-\beta))$$

$$+\alpha\log_2 p_{a_c n\beta} + (1-\alpha)\log_2(1-p_{a_c n\beta}) \tag{3.30}$$

through applying (3.26) in [88], we have

$$\begin{pmatrix} n \\ \alpha n \end{pmatrix} \le \frac{2^n C_b(\alpha)}{\sqrt{2\pi n\alpha(1-\alpha)}}, \qquad 0 < \alpha < 1 \tag{3.31}$$

also, from (3.27) we know the fact

$$\sum_{i=1}^{r} \left(\begin{array}{c} r \\ i \end{array} \right) p_i^{S_n}(1-p_i)^{n-S_n} \leq r \left(\begin{array}{c} r \\ i \end{array} \right) p_i^{S_n}(1-p_i)^{n-S_n}$$

Applying last statement in (3.23) for i = r and i = r -1, we get

$$P_r = \sum_{j=1}^{r} f_{ij} \text{ and } p_{r-1} = \sum_{j=1}^{r} \frac{r - C_r}{r} f_{ij} + \sum_{j=1}^{r} \frac{C_r}{r} f_{ij}.$$

Therefore, we proved (3.26) by using (3.19), (3.23), (3.27) and (3.30).

3.4 Experiments and Results

In this section, we discuss FTP and HTTP behaviors using both the current (fixed) and the proposed dynamic (variable) content-encoding schemes for a variety of genomic datasets with both GZIP and MFCompress compression algorithms. The examined datasets (FASTA format) were divided into two groups: actual and simulated datasets. Actual datasets were downloaded through two sources: National Center for Biotechnology Information (NCBI) [89] and University of California Santa Cruz (UCSC) [90] websites, as shown in Table 3.4 on page 57. Simulated datasets were generated via our genome generator that controls symbol repetitions to assess our encoding scheme, as can been seen in Table 3.4 on page 57.

3.4.1 Experimental Setup

This chapter compares the proposed (dynamic) content-encoding of HTTP to standard (fixed) content-encoding of HTTP, FTP, and BitTorrent protocols. Also, it used GZIP and MFCompress compression algorithms with both protocols, HTTP and FTP, to verify transmission time using both content-encoding schemes. Several datasets of sizes up to 430GB of FASTA files have been fed into these implementations to validate our content-encoding. The experiments were performed on machines that have specifications shown in Table 4.1 on page 86.

Table 3.4: Datasets used in our experiments

IDs	Source	Size(KB)	Renamed
pataa	National Center for Biotechnology Information	563,318	1
refGeneexonNuc	University of California Santa Cruz	639,183	2
envnr	National Center for Biotechnology Information	1,952,531	3
hg38	University of California Santa Cruz	11,135,899	4
patnt	National Center for Biotechnology Information	14,807,918	5
gss	National Center for Biotechnology Information	30,526,525	6
estothers	National Center for Biotechnology Information	43,632,488	7
humangenomic	National Center for Biotechnology Information	45,323,884	8
othergenomic	National Center for Biotechnology Information	346,387,292	9

Table 3.5: Experimental setup

Specifications	Details
Processor	2.4 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 4000 1024 MB
Operating System	Windows 8.1 Pro
Download	87 Mb/s
Upload	40 Mb/s
Programming Language	C# .Net
Protocols	FTP, HTTP, BitTorrent and VTTP
Dataset Sizes	550MB (1) - 340GB (9)

3.4.2 Actual Datasets Results

Our experimental results of real datasets are shown in Figure 3.10 on page 60 and other Figures and Tables at the end of this chapter through modified FTP and HTTP content-encoding with two compression algorithms (GZIP and MFCompress) as well as BitTorrent.

In order to assess the effectiveness of our dynamic encoding scheme on data transfer time, we compared the data size and transfer time of each content-encoding scheme (fixed and dynamic) of HTTP and FTP. The results show that the DVLBE decreases the size of the data that need to be transferred quickly, and the corresponding decrease in the transfer time making it faster to transfer, as shown in Figure 3.11 on page 61 and Figures and Tables at the end of this chapter.

For example, 1.20e+05 millisecond (ms) are required to transfer a compressed 550MB dataset using the traditional HTTP content-encoding with GZIP, 3.83e+04 ms via the FTP, whereas 2.02e+03 ms are required to transfer the same file via the DVLBE and 6.36e+05 ms over HTTP with MFCompress algorithm. This rate of transfer is about 98 times faster than HTTP FLBE-based, and about 95 times faster than FTP, and 99-fold faster than HTTP-FLBE with MFCompress. Also, the 30GB dataset was transferred in 7.20e+06 ms, using the HTTP FLBE and GZIP-based, 6.312e+06 ms by FTP FLBE and GZIP-based, 9.60e+07 HTTP FLBE-MFCompress-based, whereas it only took 3.26e+05 ms to transfer the file using HTTP DVLBE and GZIP-based. This is about 95-fold faster than HTTP and FTP, compared to 99-fold faster than HTTP FLBE MFCompress-based. We show results for up to 340GB. The size reduction reaches to 96%, compared to HTTP and FTP FLBE and GZIP-based of the original size as shown in Figure 5.8 on page 108. Figures and Tables at the end of this chapter provide more results about time acceleration and size reduction of actual datasets over HTTP and FTP transfer protocols FLBE and DVLBE-based with compression algorithms (GZIP and MFCompress).

In following tables, we show the acceleration time and size reduction of tested actual genomic datasets defined in Table 4.2 on page 87 at the beginning of this chapter using proposed encoding scheme over HTTP and FTP with and without compression algorithms (GZIP and MFCompress). Also, we show the speedup rate and size reduction of genomic datasets using a proposed encoding scheme in contrast to standard encoding. The results showed a significant improvement in terms of transfer time as shown in Tables 3.6, 3.7, 3.8, 3.9 and 3.10 on page 64 and table 3.11 on page 67.

3.4.3 Simulated Datasets Results

For more accuracy and further validation for the proposed encoding scheme, we performed experiments on generated datasets in FASTA format using our genome generator that is

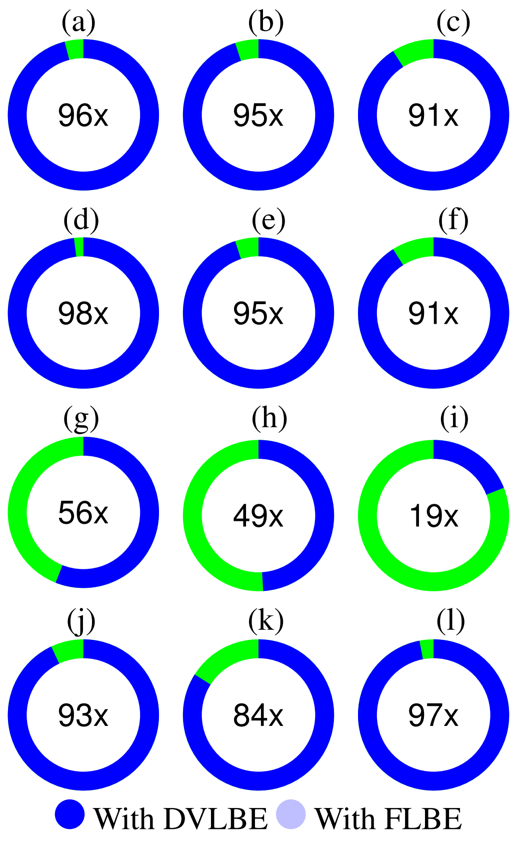
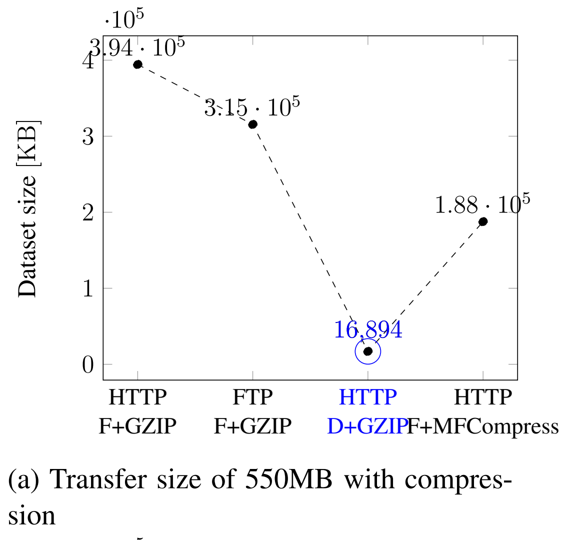
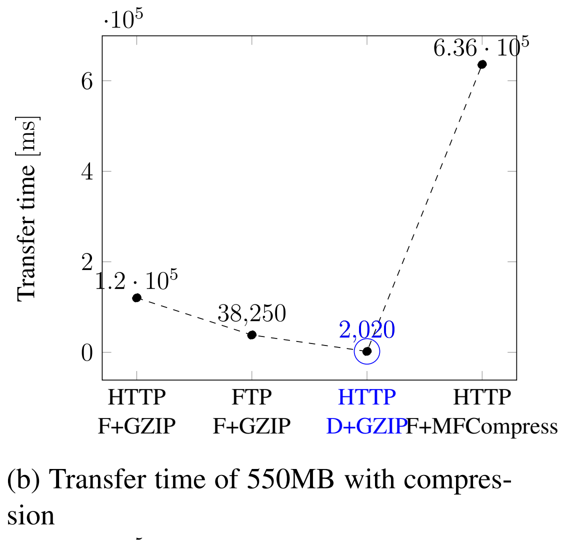


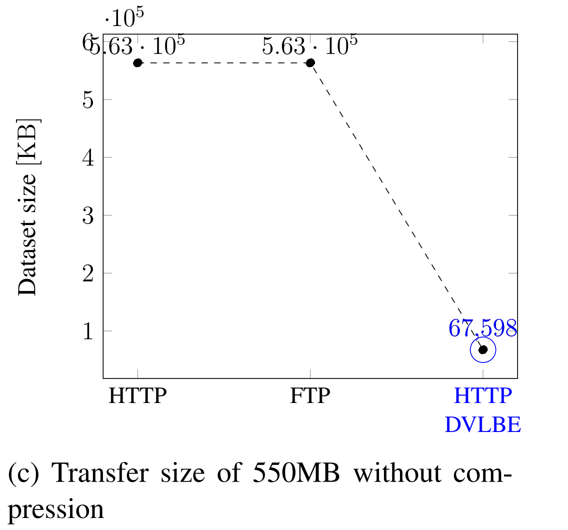
Figure 3.9: Time acceleration and size reduction of actual datasets 550MB and 430GB as defined in Table 3.4 on page 57 over multiple transfer protocols (HTTP and FTP) FLBE and DVLBE based with compression algorithms (GZIP and MFCompress). (a) HTTP FLBE and GZIP vs HTTP DVLBE and GZIP (550MB size), (b) HTTP DVLBE and GZIP vs FTP FLBE and GZIP (550MB size), (c) HTTP DVLBE and GZIP vs HTTP DVLBE and MFCompress (550MB size), (d) HTTP FLBE and GZIP vs HTTP DVLBE and GZIP (550MB time), (e) HTTP DVLBE and GZIP vs FTP FLBE and GZIP (550MB time), (f) HTTP DVLBE and GZIP vs HTTP DVLBE and MFCompress (550MB time), (g) HTTP FLBE and GZIP vs HTTP DVLBE and GZIP (340GB size), (h) HTTP DVLBE and GZIP vs FTP FLBE and GZIP (340GB size), (i) HTTP DVLBE and GZIP vs HTTP DVLBE and MFCompress (340GB size), (j) HTTP FLBE and GZIP vs HTTP DVLBE and GZIP (340GB time), (k) HTTP DVLBE and GZIP vs FTP FLBE and GZIP (340GB time), (l) HTTP DVLBE and GZIP vs HTTP DVLBE and MFCompress (340GB time)



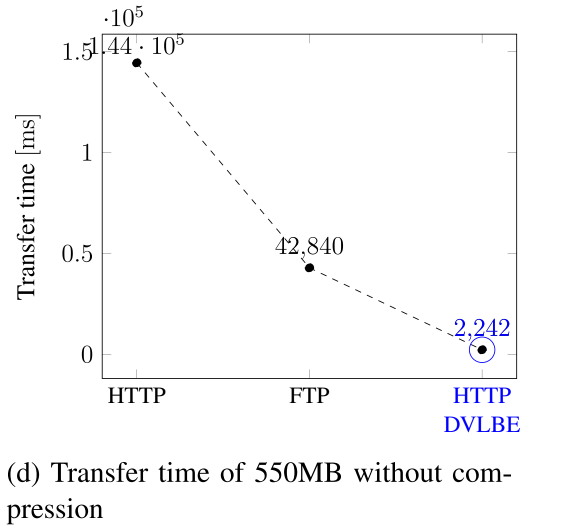
(a) Transfer size of 550MB with compression



(b) Transfer time of 550MB with compression



(c) Transfer size of 550MB without compression

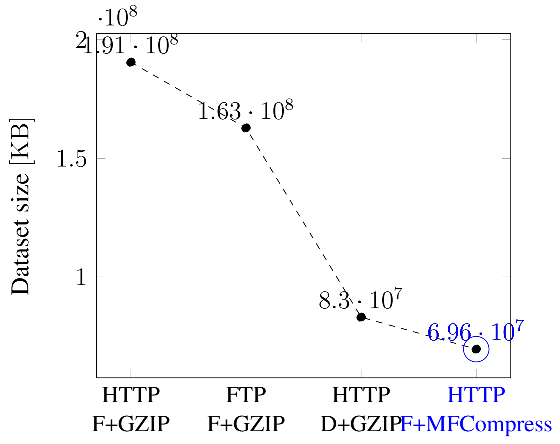


(d) Transfer time of 550MB without compression

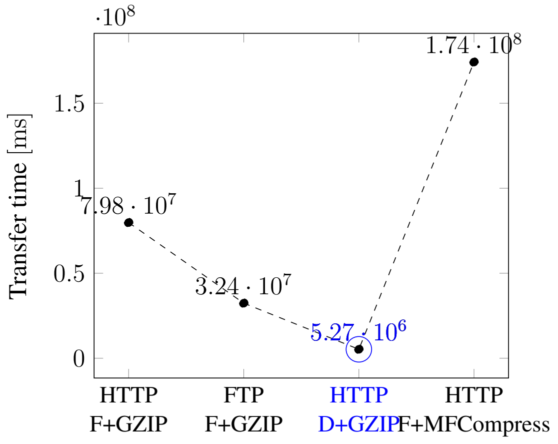
Figure 3.10: Transfer size and time of actual datasets 550MB as defined in Table 3.4 on page 57 using multiple transfer protocols (HTTP and FTP) FLBE and DVLBE based, without and with compression algorithms (GZIP and MFCompress)

shown in Table 4.2 on page 87 and Algorithm 3 on page 68. In this section, we compare and discuss the performance of HTTP and FTP using fixed-encoding comparisons to our dynamic encoding with and without compression algorithms (GZIP and MFCompress). FVLBE shows better results than FLBE, but the DVLBE showed even better results using a dynamic encoding scheme, as shown at the end of this chapter.

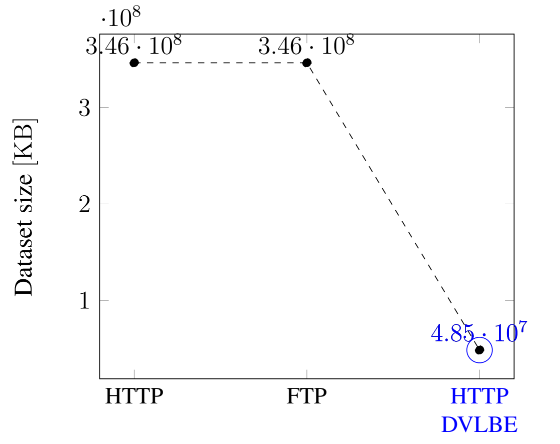
For HTTP, we utilized GZIP and MFCompress as compression algorithms, while only GZIP for FTP protocol implementation using both content-encoding schemes (fixed and dynamic) to validate our scheme. The results showed a big impact on the data transfer time when using our



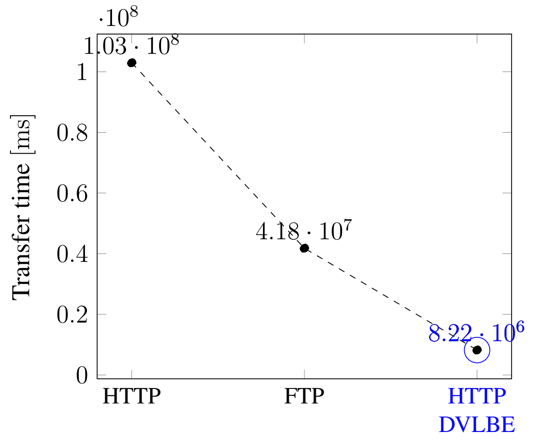
(a) Transfer size of 340GB with compression



(b) Transfer time of 340GB with compression



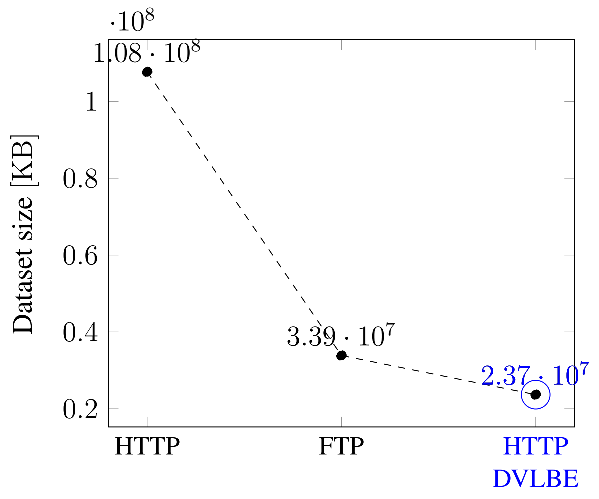
(c) Transfer size of 340GB without compression



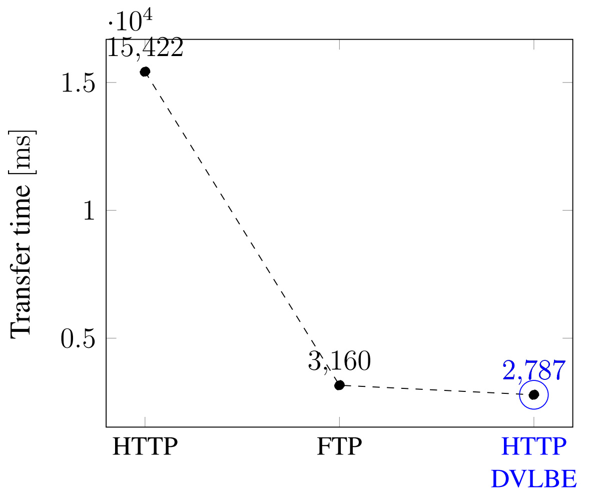
(d) Transfer time of 340GB without compression

Figure 3.11: Transfer size and time of actual datasets 340GB as defined in Table 3.4 on page 57 using multiple transfer protocols (HTTP and FTP) FLBE and DVLBE based, without and with compression algorithms (GZIP and MFCompress)

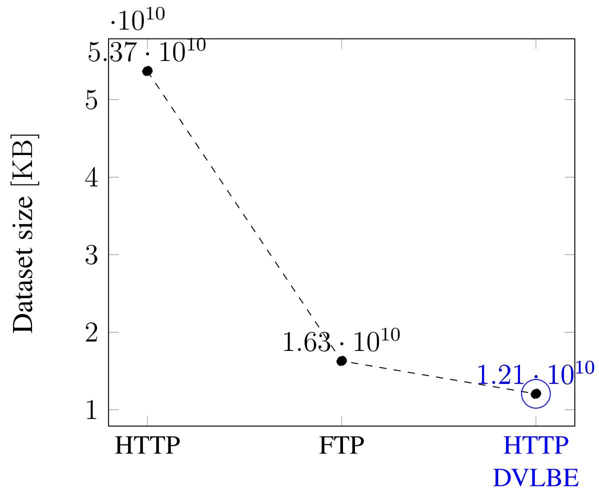
encoding scheme for the HTTP compared to fixed encoding. As expected, the transfer time for genomic datasets is always much shorter using FVLBE and DVLBE than the FLBE that is currently used in HTTP and FTP. For both encoding modes, FVLBE and DVLBE, the HTTP protocol shows significant enhancement of transfer time in contrast to FLBE due to symbol repetitions. The FVLBE and DVLBE work efficiently for files that include more symbol repetitions. Furthermore, more enhancements have been achieved by doing a dynamic sampling during encoding compared to FVLBE. However, utilizing machine learning random sample algorithm, as illustrated in section 3.3.3 on page 48, needs more strings read, and that consumes extra time before assigning the



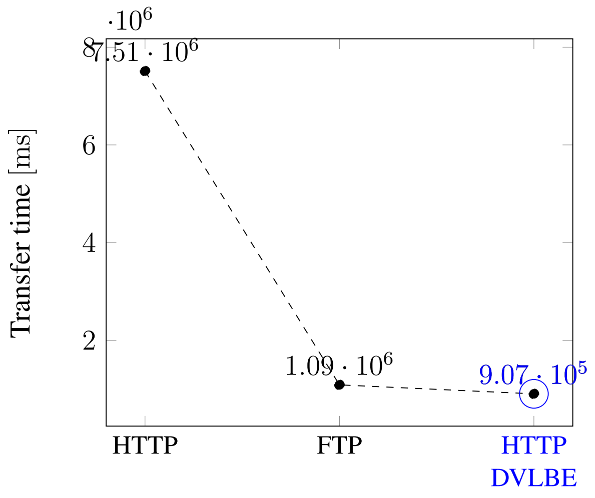
(a) Transfer size of 100MB without compression



(b) Transfer time of 100MB without compression



(c) Transfer size of 50GB without compression

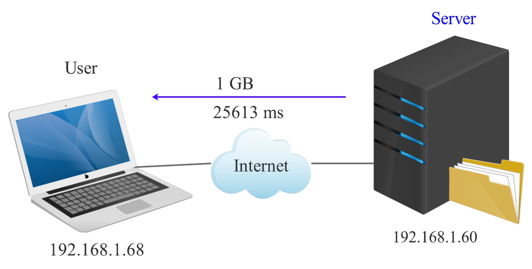


(d) Transfer time of 50GB without compression

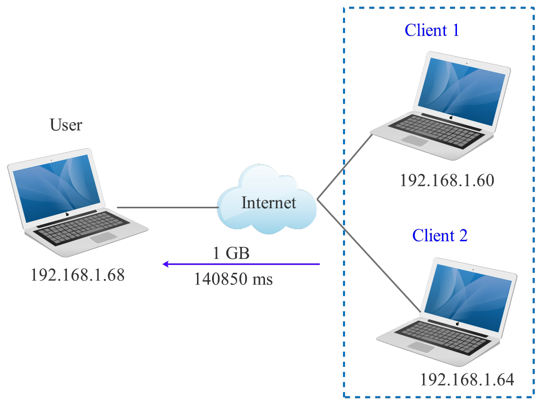
Figure 3.12: Transfer size and time of generated datasets 100MB and 50GB as defined in Table 4.2 on page 87 over multiple transfer protocols (HTTP and FTP) FLBE and DVLBE based, without using compression algorithms

shortest code for more repetition letters.

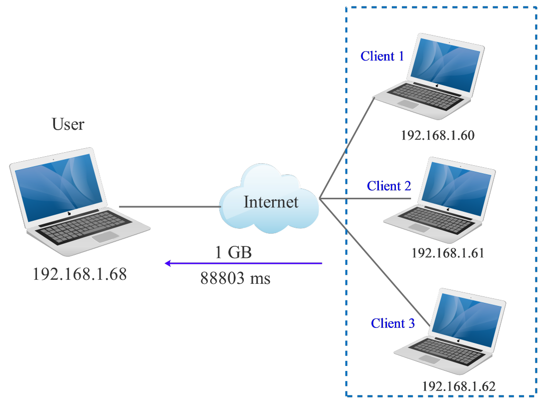
DVLBE is designed to encode datasets in minimum possible binary codewords. Therefore, DVLBE minimizes datasets significantly during the transfer process due to including more symbol *repetitions* after applying a random sample mechanism using the machine learning principle, as illustrated in section 3.3.3 on page 48. For example, the first experiment spent 2.28e+04 ms to transfer the 100M dataset with 25% frequency for the symbol A using standard encoding



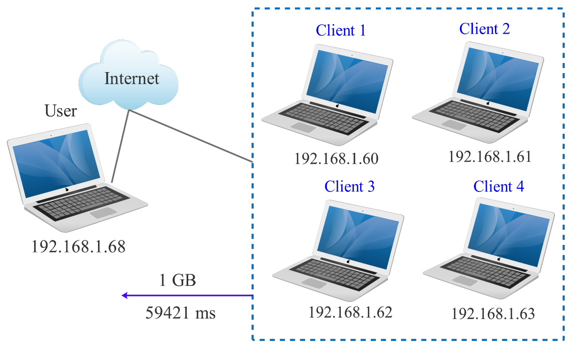
(a) Transfer time of 1GB genomic dataset using 1 machine that utilizes the proposed data minimization algorithm



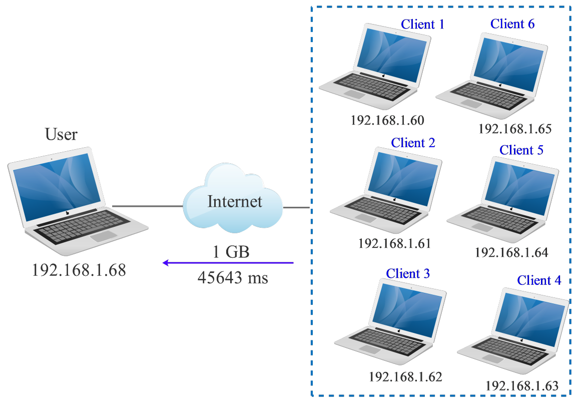
(b) Transfer time of 1GB genomic dataset using 2 machines that utilizes the standard content-encoding algorithm



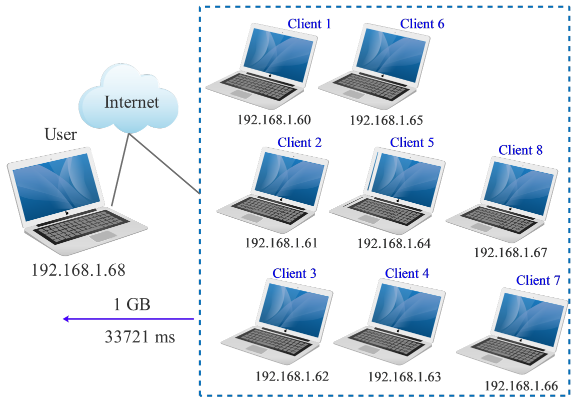
(c) Transfer time of 1GB genomic dataset using 3 machines that utilizes the standard content-encoding algorithm



(d) Transfer time of 1GB genomic dataset using 4 machines that utilizes the standard content-encoding algorithm

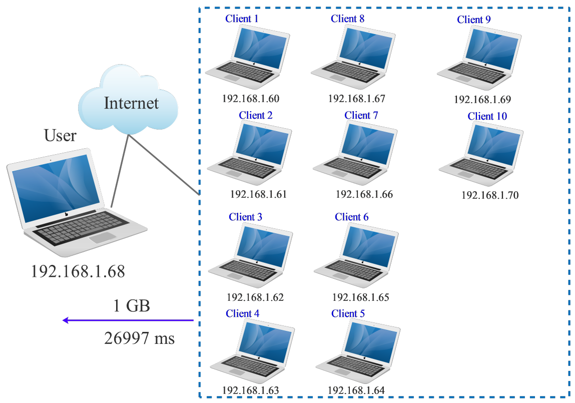


(e) Transfer time of 1GB genomic dataset using 6 machines that utilizes the standard content-encoding algorithm

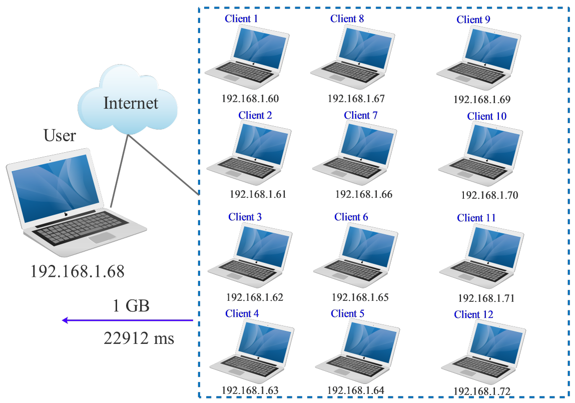


(f) Transfer time of 1GB genomic dataset using 8 machines that utilizes the standard content-encoding algorithm

Figure 3.13: Transfer time in millisecond of 1GB genomic dataset using a proposed data minimization algorithm using a single machine and a standard content-encoding algorithm using multiple (2 - 8) machines in parallel



(a) Transfer time of 1GB genomic dataset using 10 machines that utilizes the standard content-encoding algorithm



(b) Transfer time of 1GB genomic dataset using 12 machines that utilizes the standard content-encoding algorithm

Figure 3.14: Transfer time in millisecond of 1GB genomic dataset using a proposed data minimization algorithm using a single machine and a standard content-encoding algorithm using 10 and 12 machines in parallel

Table 3.6: Time acceleration comparisons of actual datasets in (ms) without compression

Dataset	HTTP	FTP	HTTP
	FLBE-based	FLBE-based	DVLBE-based
1	1.44e+05	4.28e+04	2.24e+03
2	1.68e+05	7.12e+04	7.31e+03
3	6.10e+05	1.66e+05	1.11e+04
4	3.33e+06	1.24e+06	8.08e+04
5	5.11e+06	3.22e+06	1.98e+05
6	8.86e+06	7.58e+06	4.90e+05
7	1.44e+07	1.06e+07	7.77e+05
8	2.23e+07	1.30e+07	8.01e+05
9	1.03e+08	4.18e+07	8.22e+06

over HTTP, 1.27e+04 ms via the FTP, compared to 3.25e+03 ms to transfer the same file via the DVLBE with 7-fold faster than standard encoding.

The behaviors were impacted by 2 factors: number and repetition of symbols that occurred during the design of this content-encoding scheme. Therefore, our transfer protocol that uses DVLBE of machine learning random sample, as illustrated in section 3.3.3 on page 48, pro-

Table 3.7: Time acceleration comparisons of actual datasets in (ms) with compression

Dataset	HTTP	FTP	HTTP	HTTP
	FLBE-GZIP	FLBE-GZIP	DVLBE-GZIP	FLBE-MFCompress
1	1.20e+05	3.83e+04	2.02e+03	6.36e+05
2	1.20e+05	5.89e+04	4.57e+03	6.82e+05
3	4.21e+05	1.42e+05	5.85e+03	3.73e+06
4	2.52e+06	1.15e+06	4.25e+04	N/A
5	3.60e+06	2.78e+06	1.52e+05	3.75e+05
6	7.20e+06	6.32e+06	3.26e+05	9.60e+07
7	1.08e+07	8.62e+06	4.57e+05	2.24e+08
8	1.80e+07	1.05e+07	5.72e+05	2.46e+07
9	7.98e+07	3.24e+07	5.27e+06	1.74e+08

Table 3.8: Size reduction comparisons of actual datasets in (KB) without compression

Dataset	HTTP	FTP	HTTP
	FLBE-based	FLBE-based	DVLBE-based
1	5.63e+05	5.63e+05	6.76e+04
2	6.39e+05	6.39e+05	9.59e+04
3	1.95e+06	1.95e+06	3.32e+05
4	1.11e+07	1.11e+07	1.45e+06
5	1.48e+07	1.48e+07	2.81e+06
6	3.05e+07	3.05e+07	4.58e+06
7	4.36e+07	4.36e+07	5.67e+06
8	4.53e+07	4.53e+07	7.25e+06
9	3.46e+08	3.46e+08	4.85e+07

vides a better performance for big genomic data. Also, this protocol presents an ideal solution to transfer all genomic datasets that contain different symbol repetitions with an average acceleration of 99-fold compared to the traditional HTTP. The time acceleration and size reduction for all FASTA files in Table 4.2 on page 87 using all mentioned encoding schemes. This encoding scheme, when compared to a standard scheme, features improved reliability, scalability, performance and security. The reliability relies in its ability to use both encoding (genomics and universal) schemes

Table 3.9: Size reduction comparisons of actual datasets in (KB) with compression

Dataset	HTTP	FTP	HTTP	HTTP
	FLBE-GZIP	FLBE-GZIP	DVLBE-GZIP	FLBE-MFCompress
1	3.94e+05	3.15e+05	1.69e+04	1.88e+05
2	3.96e+05	3.13e+05	6.64e+04	9.31e+04
3	1.35e+06	8.79e+05	8.78e+04	6.67e+05
4	6.24e+06	5.79e+06	7.96e+05	N/A
5	8.74e+06	6.22e+06	2.97e+06	2.34e+06
6	1.56e+07	1.34e+07	6.42e+06	5.39e+06
7	2.49e+07	1.79e+07	8.75e+06	6.47e+06
8	2.81e+07	2.31e+07	1.11e+07	9.08e+06
9	1.91e+08	1.63e+08	8.30e+07	6.96e+07

Table 3.10: Size reduction rates of actual datasets with compression (D refers to DVLBE and F refers to FLBE)

Dataset	HTTP	HTTP	HTTP vs FTP
	D-GZIP vs F-GZIP	D-GZIP vs F-MFCompress	D-GZIP vs F-GZIP
1	96%	91%	95%
2	83%	29%	79%
3	93%	87%	90%
4	87%	N/A%	86%
5	66%	127%	52%
6	59%	119%	52%
7	65%	135%	51%
8	61%	122%	52%
9	56%	119%	49%

when assigning a codeword for each symbol. The scalability feature is present by enabling the protocol to update the binary representation of each alphabet symbol according to the machine learning random sample, as illustrated in section 3.3.3 on page 48. This encoding scheme indicates better performance via reducing the dataset during data transfer, which, in turn, reduces the network traffic. The security of this protocol stems from utilizing the TCP protocol in the transport

Table 3.11: Time acceleration rates of actual datasets with compression (D refers to DVLBE and F refers to FLBE)

Dataset	HTTP	HTTP	HTTP vs FTP
	D-GZIP vs F-GZIP	D-GZIP vs F-MFCompress	D-GZIP vs F-GZIP
1	98x	99x	95x
2	96x	99x	92x
3	99x	99x	96x
4	98x	N/A	96x
5	96x	99x	95x
6	95x	99x	95x
7	96x	99x	95x
8	97x	98x	95x
9	93x	97x	84x

layer of the TCP/IP model.

In order to compare the results of the proposed content-encoding of HTTP with BitTorrent protocol, we implemented the latter approach, as well. The results are shown in Figure 3.13 on page 63 for a 1GB FASTA file that was downloaded from the NCBI website. Only 1 machine (server) is used to transfer the same file using DVLBE, while *n* machines are used with the same file using BitTorrent protocol FLBE-based. As expected, with the increasing number of machines, the time to transfer decreases sharply over BitTorrent. It can also be observed that 1 machine using HTTP DVLBE-based takes the same time to transfer 1GB of the genomic dataset using 10 parallel machines over BitTorrent. This is due to massive reduction in size due to our encoding strategy. Also, note that employing HTTP DVLBE-based for multiple machines will massively decrease the time needed to transfer a file of a given size.

We generated 21 genomic datasets with variety of symbol occurrences (25%, 35% and 50%) using our genomic generator shown in algorithm 3 on page 68 to monitor and verify our encoding scheme. Thus, we manually configured 6 possible encoding trees to assess our dynamic encoding performance. Theses encoding possibilities were 0345, 5430, 3504, 4053, 0534 and 0543 represent fvlbe1, fvlbe2, fvlbe3, fvlbe4, fvlbe5, fvlbe6 respectively, fvlbe refers to fixed variable-length binary encoding.

DVLBE refers to dynamic variable length binary encoding and flbe denotes to fixed length binary encoding. The results show a significant size reduction during transferring the data that led to shortening the transfer time as can be seen in Tables 3.12 and 3.13 on page 69 and figures 3.15 and 3.16 on page 71 and figures 3.17 and 3.18 on page 74.

Algorithm 3 A genomic generator algorithm

```
 1: procedure GENERATE
 2:     FileSize ← Get Genomic File Size .
 3:     TotalSymoblsToBeGenerated ← FileSize .
 4:     Mode ← Get Generation Mode.
 5:     if Mode = Manually then
 6:         Symbol ← Get Genomic Symbol to control its ← frequency.
 7:         Ratio ← Get the Genomic Symbol's percentage .
 8:         SymbolGenerate(OutputFile,Symbol, ← Ratio * FileSize).
 9:         TotalSymoblsToBeGenerated ← FileSize *(1-Ratio).
10:     end if
11:     RandomlyInitArray(SymbolsArray).  ▷ SymbolsArray will be filled with genomic symbols
        randomly, the array size is calculated randomly too
12:     while TotalSymoblsToBeGenerated > 0 do
13:         ArrayIndex ← Random.GetNumber % ← SymbolsArray.Length.     ▷ Pick an element
        from SymbolsArray randomly
14:         GenomicSybmol ← SymbolsArray[ArrayIndex].
15:         OutputFile.write(GenomicSybmol).
16:         TotalSymoblsToBeGenerated ← TotalSymoblsToBeGenerated -1 .
17:     end while
18: end procedure
```

3.5 Conclusions

In this chapter, we implemented a novel machine learning-based data minimization algorithm to be integrated with transfer protocols to reduce the size of big genomic datasets during the transfer phase, and then to transfer the data securely in a shorter time. The implementation results illustrate that the proposed data minimization algorithm is capable of reducing the transfer time 99-fold, compared to the standard content-encoding of HTTP, and 96-fold compared to FTP on tested datasets. We used GZIP and MFCompress algorithms as optional compression algorithms

Table 3.12: Size reduction comparisons of simulated datasets in (kB)

Dataset	HTTP	FTP	HTTP	HTTP vs HTTP	HTTP vs FTP
	FLBE + GZIP	FLBE + GZIP	DVLBE + GZIP	DVLBE vs FLBE	DVLBE vs FLBE
100MB[25]	1.078e+08	1.078e+08	2.906e+07	73%	73%
100MB[35]	1.075e+08	1.075e+08	2.703e+07	75%	75%
100MB[50]	1.077e+08	1.077e+08	2.367e+07	78%	78%
500MB[25]	5.369e+08	5.369e+08	1.473e+08	73%	73%
500MB[35]	5.372e+08	5.372e+08	1.352e+08	75%	75%
500MB[50]	5.369e+08	5.369e+08	1.217e+08	77%	77%
1GB[25]	1.074e+09	1.074e+09	2.928e+08	73%	73%
1GB[35]	1.074e+09	1.074e+09	2.769e+08	74%	74%
1GB[50]	1.074e+09	1.074e+09	2.430e+08	77%	77%
5GB[25]	5.369e+09	5.369e+09	1.466e+09	73%	73%
5GB[35]	5.369e+09	5.369e+09	1.393e+09	74%	74%
5GB[50]	5.369e+09	5.369e+09	1.195e+09	77%	77%
10GB[25]	1.074e+10	1.074e+10	2.922e+09	73%	73%
10GB[35]	1.074e+10	1.074e+10	2.763e+09	74%	74%
10GB[50]	1.074e+10	1.074e+10	2.433e+09	77%	77%
50GB[25]	5.369e+10	5.369e+10	1.486e+10	72%	72%
50GB[35]	5.369e+10	5.369e+10	1.412e+10	74%	74%
50GB[50]	5.369e+10	5.369e+10	1.205e+10	78%	78%

beside our data minimization algorithm to assess how the transfer protocol behaves in terms of transfer time and size. Also, we showed that our data minimization algorithm provides the best size reduction, reduces transfer time, and securely transfers big genomic datasets.

Our proposed data minimization mechanism relies on a machine learning-based random sampling method while encoding the data during data transfer, and then transfers the data securely in a shortened time, as illustrated in section 3.3.3 on page 48. We showed that the data size can be significantly reduced by our adaptive encoding, compared to a standard content-encoding scheme, with and without compression algorithms, as well. Also, we implemented a genomic dataset generator of FASTA file format to verify the performance of our current data minimization scheme and compared it to the standard as well as our previous content-encoding schemes. Our proposed genome generator allowed us to control the repetition in the data, which was instrumen-

Table 3.13: Time acceleration comparisons of simulated datasets in (ms)

Dataset	HTTP	FTP	HTTP	HTTP vs HTTP	HTTP vs FTP
	FLBE + GZIP	FLBE + GZIP	DVLBE + GZIP	DVLBE vs FLBE	DVLBE vs FLBE
100MB[25]	2.277e+04	1.265e+04	3.249e+03	7.01x	3.89x
100MB[35]	1.997e+04	8.681e+03	2.710e+03	7.37x	3.20x
100MB[50]	1.542e+04	9.639e+03	2.787e+03	5.53x	3.46x
500MB[25]	9.534e+04	5.417e+04	1.133e+04	8.41x	4.78x
500MB[35]	8.502e+04	4.383e+04	1.049e+04	8.11x	4.18x
500MB[50]	7.595e+04	5.238e+04	9.205e+03	8.25x	5.69x
1GB[25]	1.881e+05	8.956e+04	1.837e+04	10.24x	4.87x
1GB[35]	1.665e+05	6.938e+04	1.858e+04	8.96x	3.73x
1GB[50]	1.289e+05	6.894e+04	1.731e+04	7.45x	3.98x
5GB[25]	9.830e+05	5.922e+05	1.152e+05	8.53x	5.14x
5GB[35]	8.448e+05	3.840e+05	9.681e+04	8.73x	3.97x
5GB[50]	6.662e+05	2.221e+05	7.688e+04	8.67x	2.89x
10GB[25]	1.881e+06	6.967e+05	1.716e+05	10.96x	4.06x
10GB[35]	1.670e+06	8.433e+05	1.902e+05	8.78x	4.43x
10GB[50]	1.367e+06	4.712e+05	1.787e+05	7.65x	2.64x
50GB[25]	1.005e+07	4.189e+06	1.149e+06	8.75x	3.64x
50GB[35]	8.793e+06	4.677e+06	1.073e+06	8.19x	4.36x
50GB[50]	7.511e+06	3.266e+06	9.066e+05	8.29x	3.60x

tal in assessing the performance of our data minimization algorithm. The tested encoding schemes (FLBE, FVLBE and DVLBE) were implemented over HTTP, FTP and BitTorrent protocols, with and without compression algorithms.

Our experiments indicated that machine learning-based random sampling content-encoding performs much better than the current and common use of the transfer protocol content-encoding scheme through assigning short codewords for the dataset characters. We conclude that proposed data minimization algorithm provides the best performance among current content-encoding approaches for big genomic datasets.

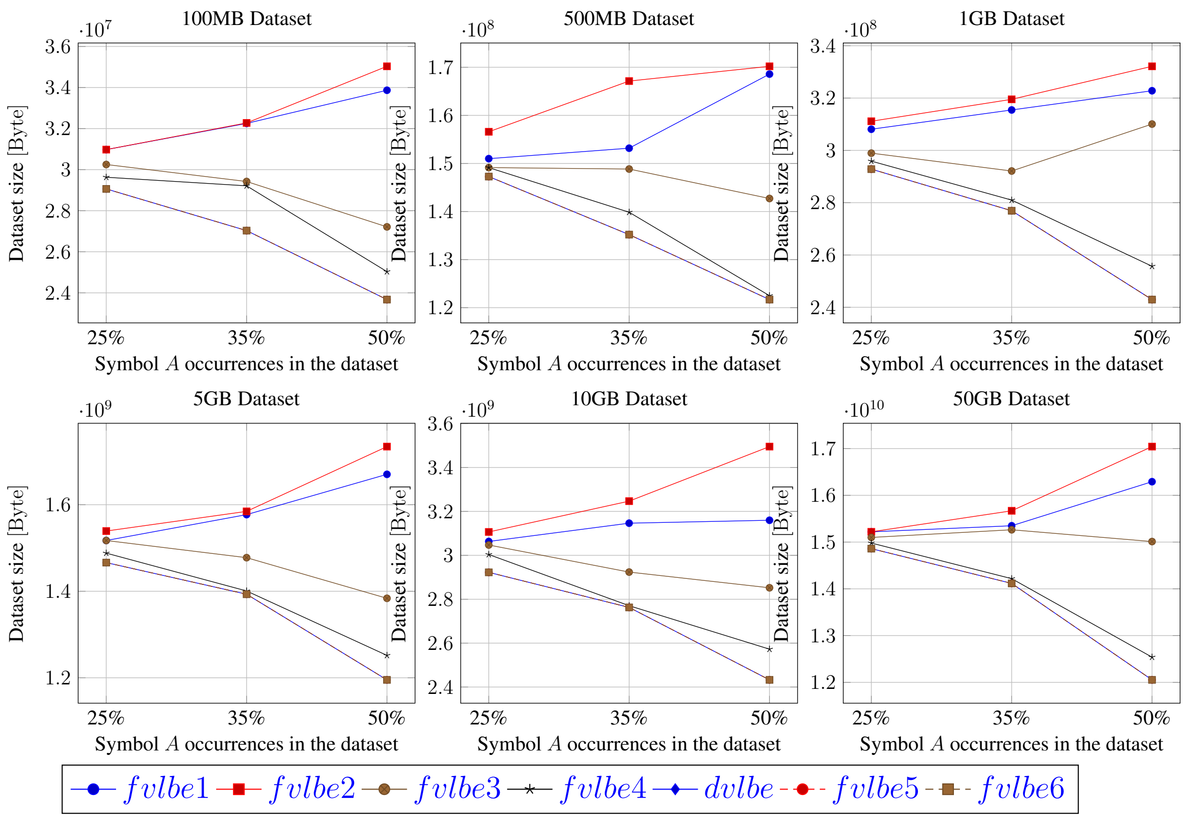


Figure 3.15: Size comparisons for multiple dataset using alternative scheme (FVLBE and DVLBE sets)

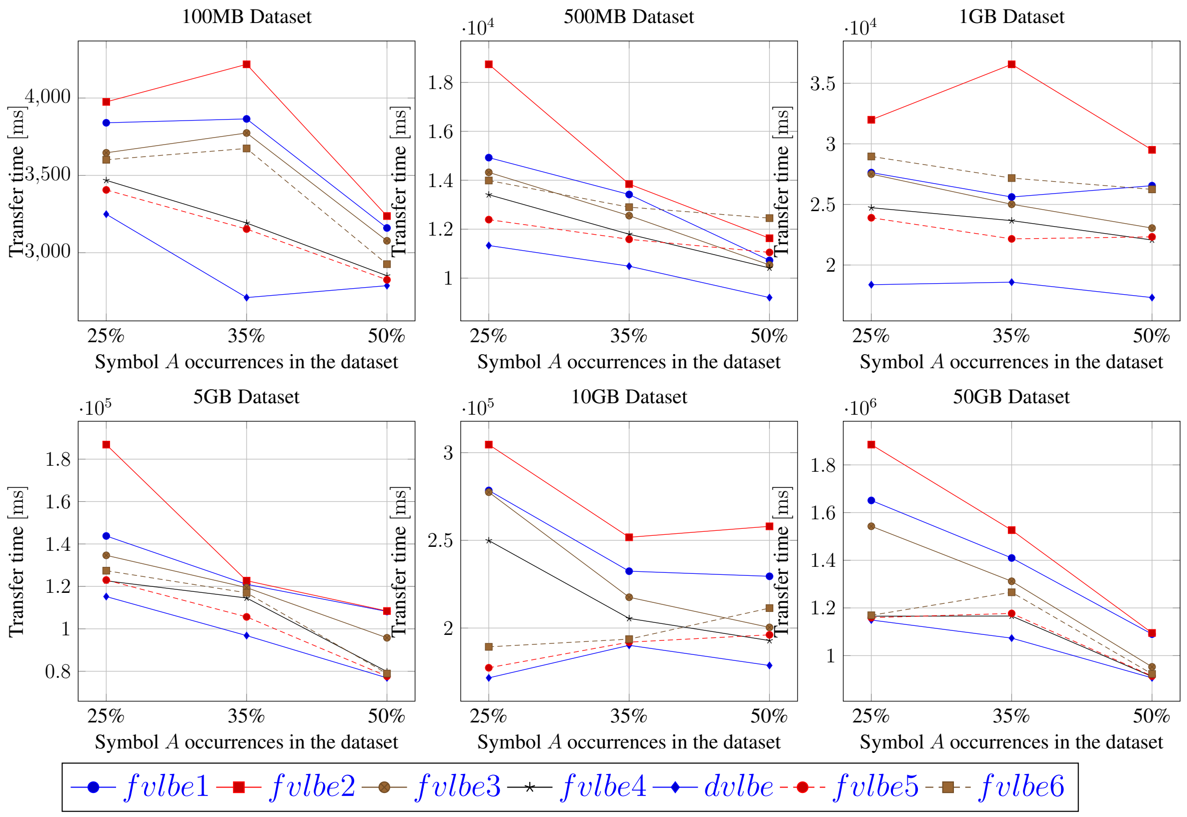


Figure 3.16: Time comparisons for multiple dataset using alternative scheme (FVLBE and DVLBE sets)

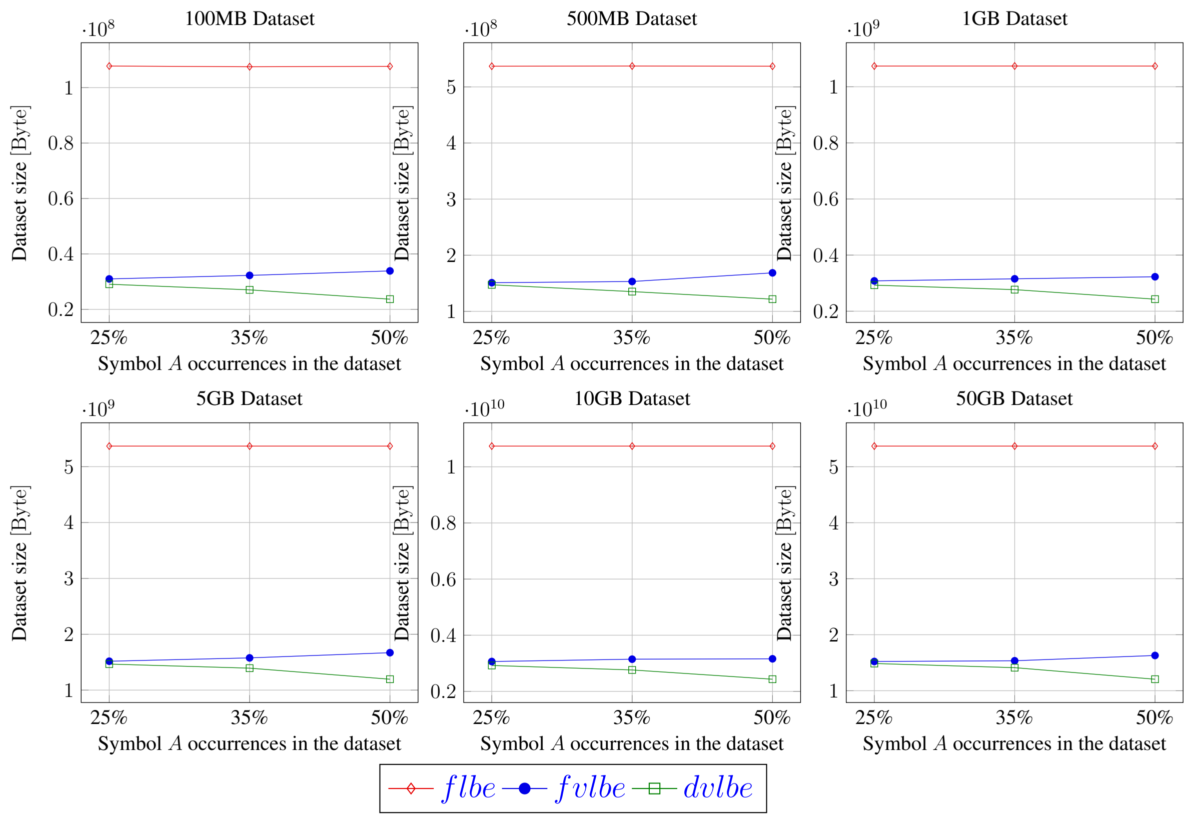


Figure 3.17: Size comparisons for multiple dataset using FLBE, FVLBE and DVLBE schemes

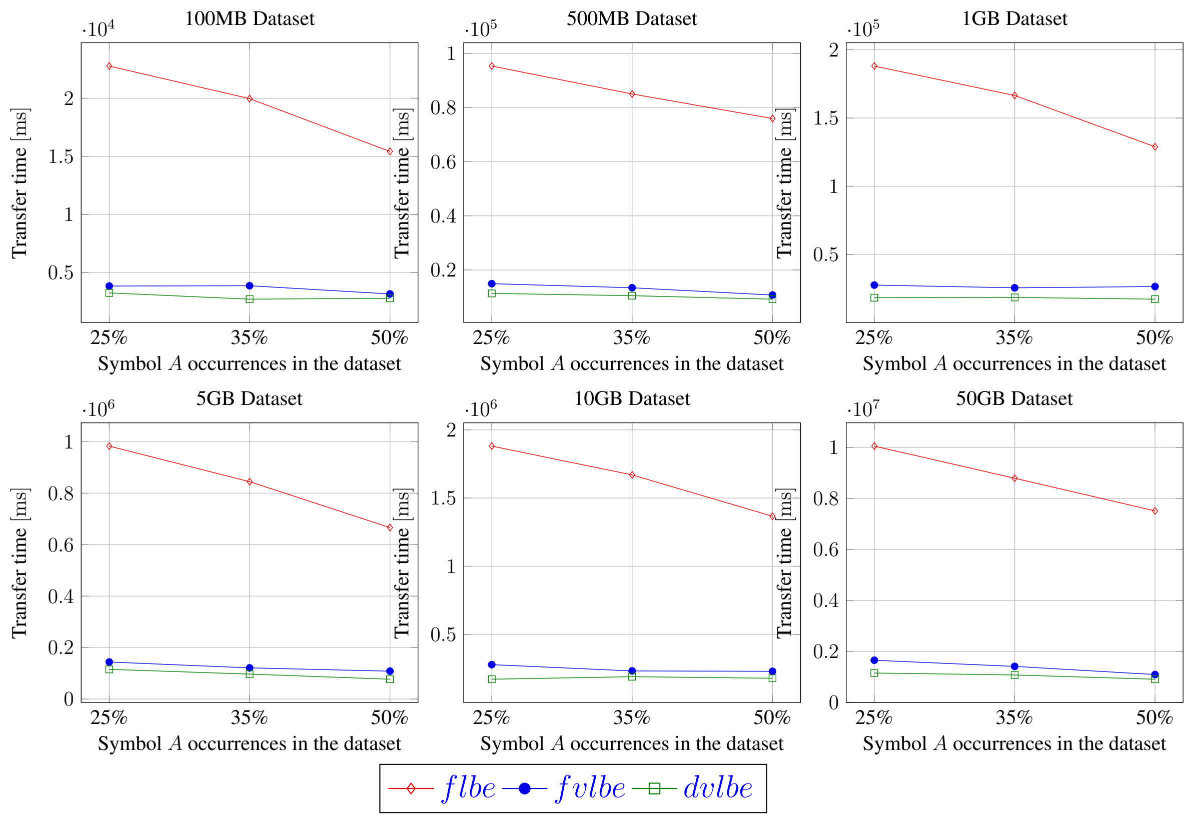


Figure 3.18: Time comparisons for multiple dataset using FLBE, FVLBE and DVLBE schemes

CHAPTER 4

A FOURIER-BASED DATA MINIMIZATION ALGORITHM FOR FAST AND SECURE TRANSFER OF BIG GENOMIC DATASETS

4.1 Introduction

DNA sequencing is a critical biological technique used by laboratories to investigate diseases and genetic illnesses. Also, it has been defined as a mechanism to make or produce proteins and organic enzymes that determine bodily functions. DNA sequencing provides enhanced treatment methods for genetic diseases which result from mutated DNA that causes a change in the protein product. Additionally, DNA sequencing is the only practical resource in understanding ethnicity and ancestry, as well as detecting specific genetic patterns. Fortunately, the advancement of biomedical equipment and the growth of high-throughput DNA sequencing resulted in an increase in the sequencing operations. Consequently, DNA sequencing has generated big genomic datasets that face several challenges such as storage, manipulation, analysis, visualization, and sharing.

The use of the higher throughput and less expensive equipment such as the next-generation sequencing (NGS) [91], enables more researchers to sequence more organisms that enhance quality of life. For example, less than $1,000 is the cost for sequencing a single human genome [92]. Also, biologists anticipate that the sequencing cost will continue to decrease, a concept based on a variant of Moore's law trajectory: costs decrease over time. For example, the 1000-Genomes project [93] generated a greater amount of data using the NGS in 6 months than what was generated in 21 years by NCBI Genbank [94].

Also, many genome related projects, such as the 1000-Genomes project [93] and the international cancer sequencing consortium [95], suggest using the cloud as an infrastructure to solve the store and analysis challenges [96], [97], [98]. However, the transfer and share of the genomic datasets between biological laboratories and from/to the cloud represents an ongoing bottleneck because of the amount of data and the limitations of the network bandwidth [99]. Therefore, it is critical to develop new ways to minimize the data size. This chapter presents a new real-time data minimization mechanism of big genomic datasets to shorten the transfer time in a more secure manner, despite the potential occurrence of a data breach. Our method involves the application of the random sampling of Fourier transform theory to the real-time generated big genomic datasets of both formats: FASTA and FASTQ and assigns the lowest possible codeword to the most frequent characters of the datasets.

Our results indicate that the proposed data minimization algorithm is up to 79% of FASTA datasets' size reduction, with 98-fold faster and more secure than the standard data-encoding method. Also, the results show up to 45% of FASTQ datasets' size reduction with 57-fold faster than the standard data-encoding approach. Based on our results, we conclude that the proposed data minimization algorithm provides the best performance among current data-encoding approaches for big real-time generated genomic datasets.

4.1.1 Contribution

This chapter discusses the design and implementation of a novel data minimization algorithm for the real-time generated big genomic datasets that relies on a combination of the random sampling of Fourier transform [100] and variable-length binary-encoding [50]. This work aims to minimize and secure big genomic datasets during the data transfer phase over the networks, either via wire or wireless. We assert that creating a new minimization mechanism for big genomic datasets can significantly shorten the transfer time and secure the data at the same time by changing the data-encoding of the datasets multiple times during the transfer phase. Consequently, the transfer time will decrease tremendously, in addition to protecting the data against a potential breach by changing the codewords of the genomic symbols frequently. Therefore, we can ensure that the data will transfer faster and in a more secure way.

4.1.2 Motivation

It is not a minor challenge to transfer big data in particular genomic datasets faster than the current transfer time because all data transfer protocols, such as Hyper Text Transfer Protocol (HTTP) [19] and File Transfer Protocol (FTP) [37], use the standard content-encoding schemes, such as the American Standard Code for Information Interchange (ASCII) [101]. Also, the decrease in the costs of DNA sequencing due to the advancement of the DNA sequence techniques and equipment, encourages biologists to extend their research and produce increasingly greater numbers of genomic datasets that need to be manipulated and transferred between various biology laboratories. Also, the use of DNA sequencing in the most critical areas such as criminal investigations, genotyping and determination of disease-relevant genes or agents causing diseases, mutation analysis, screening of single nucleotide polymorphisms (SNPs), detection of chromosome abnormalities [102], and global determination of post-translational modification [103] have led to the high demand to transfer and share the data faster than the current transfer time. However, shortening the transfer time of big datasets is complicated because all Internet browsers use the standard content-encoding schemes in terms of symbol lengths, such as 8-bit, 9-bit, 16-bit, 32-bit codewords.

Data minimization and privacy are very important to both users and service providers, especially for remote healthcare services. Many solutions, such as the cloud, have been developed to address the challenges of remote diagnostic devices. However, data minimization and privacy challenges have not been addressed at the same level, mainly due to compatibility issues. As a result, scientists are motivated to navigate and search for new methods to transfer big genomic data more efficiently and in a fully secured environment.

Current data transfer protocols are not suitable for the increased growth of big data and cloud-based services such as remote healthcare diagnostics due to the use of transfer protocols that belong to different vendors. Observing these facts, we take advantage of the nature of the genomic symbols to implement a more efficient content-encoding algorithm to minimize the data amount during the transfer phase.

4.1.3 Chapter Goals and Organization

The purpose of this chapter is to design and implement a novel data minimization algorithm to decrease the required time to transfer big genomic datasets over networks. In addition, we increased security in the event of a data breach. Moreover, it will introduce a generic concept that can be used by other limited symbols alphabet of the cloud-based applications to secure data that exchange remotely. The contributions of this chapter are outlined as follows:

- Summarizes the data minimization and data encoding methods with quick fundamentals, sans the need to search through the details presented in the standards' specifications.
- Provides an overview of the challenges of the big genomic datasets in terms of transfer time with extra protection if data breach occurs.
- Presents the need for better data minimization techniques to provide faster data transfer, especially cloud-based services.

The remainder of this chapter is organized as follows: Section 4.2 presents model description and formulation. Section 4.3 presents the experiments and results of the proposed data minimization algorithm. Section 4.4 presents our conclusions.

4.2 Model Description and Formulation

4.2.1 Model Description

This subsection presents our implementation of data minimization algorithm for the generation of big real-time genomic DNA datasets using DNA-sequencing equipment such as NGS. The generated datasets can have many formats, but the common ones are FASTA [104] and FASTQ [105] formats, as illustrated in Figure 4.2 on page 80. This work has been done to minimize the size of big genomic datasets and then shorten the transfer time. Also, this work adds an extra security layer via changing the symbol codewords several times for each dataset during the transfer phase. This model ensures the assignment of the lowest possible codewords for more

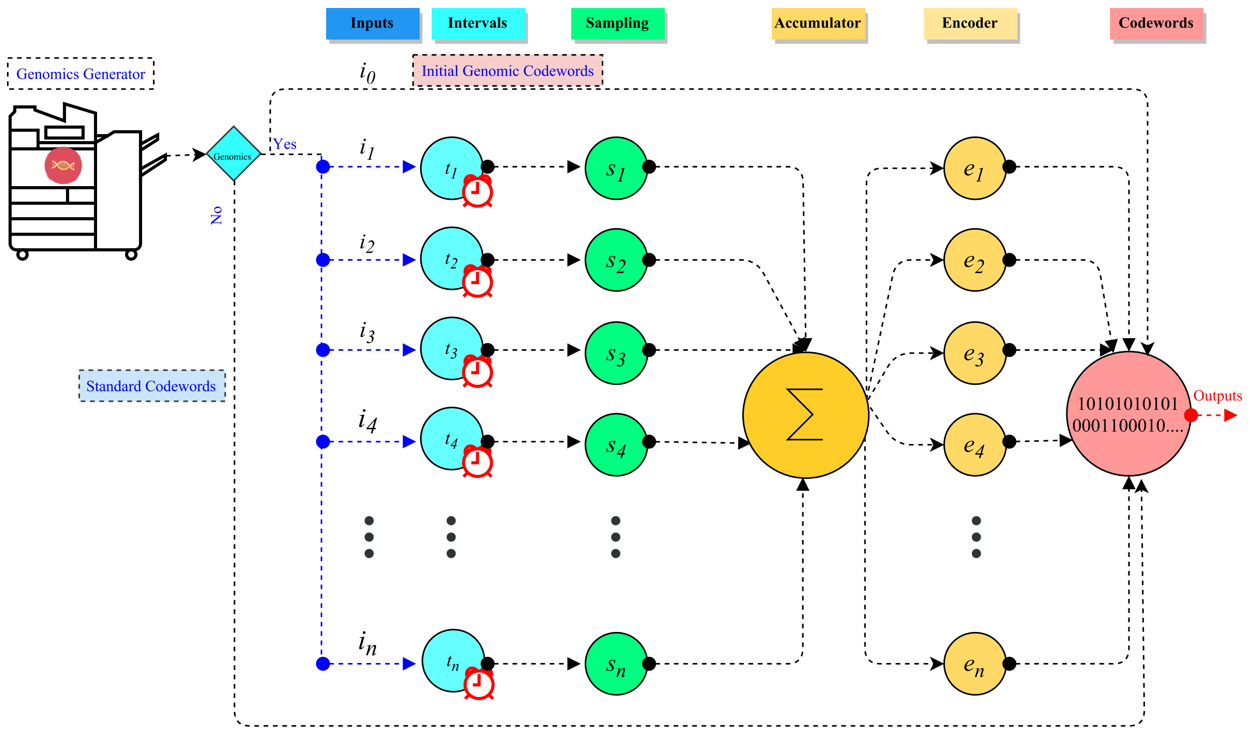
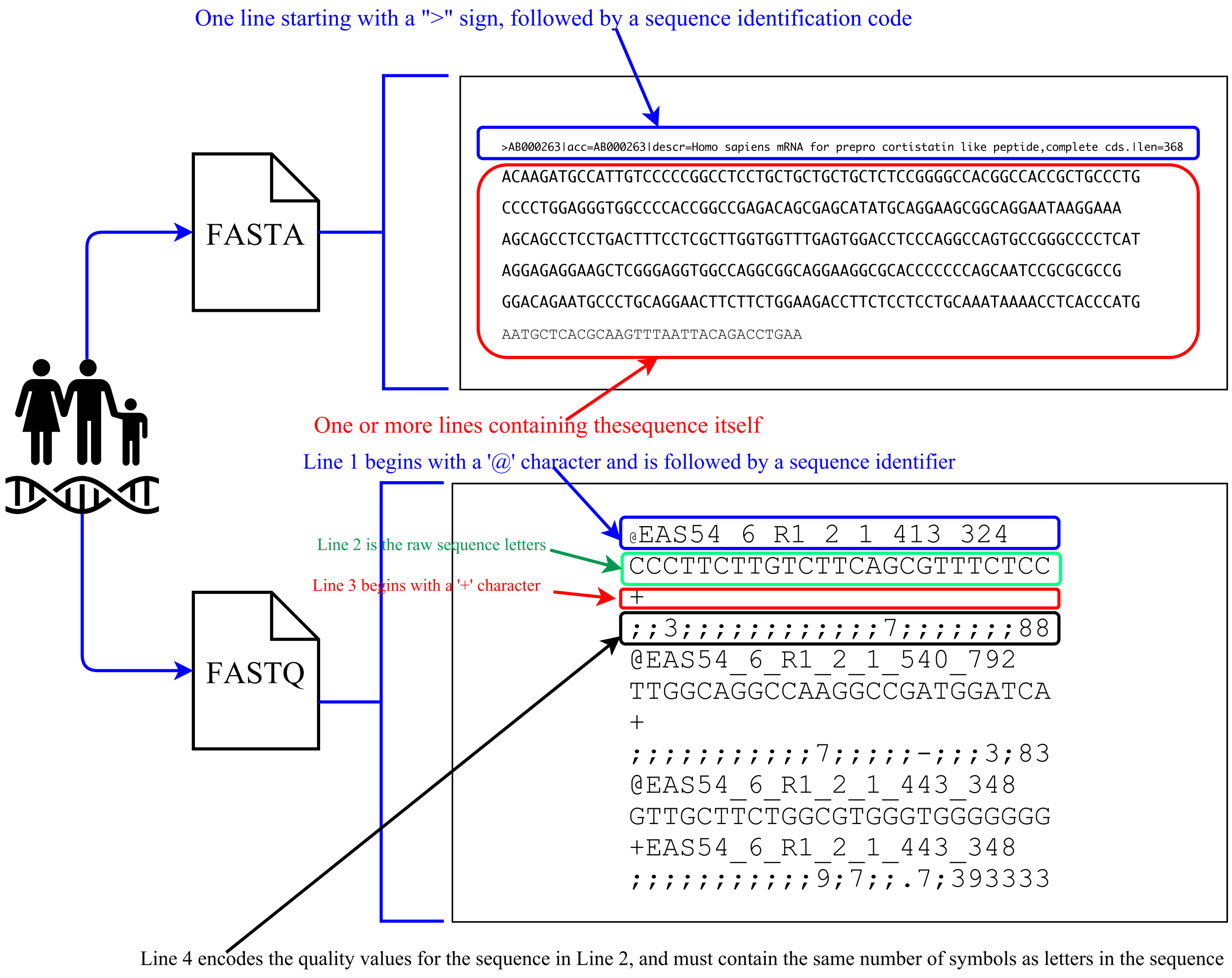


Figure 4.1: The proposed encoding system framework

symbol occurrences via dividing the datasets into parts based on time and by utilizing a random sampling of Fourier transform theory, as illustrated in Figure 4.1 on page 79 and Algorithm 4 on page 81.

Two encoding schemes are used in this model: standard and modified. The standard one uses the dataset headers and non-DNA symbols in the dataset bodies. The modified encoding scheme starts with an initial codeword, and then updates frequently via running a time-based sampling using a Fourier transform theory. The initial codewords of the proposed encoding codewords, such as those shown in Figure 2.2 (c) on page 18, periodically updates codewords via random sample of Fourier theory, as illustrated in Figure 4.1 on page 79 and Algorithm 4 on page 81.

The data-encoding switches between the standard and proposed ones by sending 25-byte of 1's from the sender to the receiver, indicating that the alternative codewords will be used from this point on, and continuously, until receiving another data-encoding scheme. Therefore, this model encodes the contents using two main codeword tables: fixed and variable (changes many


One line starting with a ">" sign, followed by a sequence identification code
FASTA
FASTQ
One or more lines containing thesequence itself
Line 1 begins with a '@' character and is followed by a sequence identifier
Line 2 is the raw sequence letters
Line 3 begins with a '+' character
Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence

```
>AB000263|acc=AB000263|descr=Homo sapiens mRNA for prepro cortistatin like peptide,complete cds.|len=368
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTG
CCCCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAA
AGCAGCCTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCAT
AGGAGAGGAAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCG
GGACAGAATGCCCTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATG
AATGCTCACGCAAGTTTAATTACAGACCTGAA
```

```
@EAS54 6 R1 2 1 413 324
CCCTTCTTGTCTTCAGCGTTTCTCC
+
;;3;;;;;;;;;;;;7;;;;;;;88
@EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
+
;;;;;;;;;;;7;;;;;-;;;3;83
@EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
+EAS54_6_R1_2_1_443_348
;;;;;;;;;;;9;7;;.7;393333
```

Figure 4.2: The FASTA and FASTQ format components

times), as shown in Figure 4.1 on page 79. The proposed data-encoding scheme of the genomic dataset benefits from being in the alphabet that can be encoded in four unique decipherable code-words, i.e. [0, 11, 100, 101] or simply [0, 3, 4, 5]. The result of the sampling phase adds to the encoding accumulator in an array form, and then calculates symbol repetitions and creates the new encoding tree that is used in the data-encoding phase.

We utilize a binary tree as a structure to represent our genomic data-encoding because it is faster to search, avoids duplicate values, and facilitates decoding at the receiver side. Also, the use of a binary tree, as a structure, gives the programmer special flexibility because it offers one of the two paths to follow and cuts the search time in half, thereby increasing process throughput.

Algorithm 4 Proposed data-encoding algorithm

```
1: procedure DATA-ENCODING
2:     Proposed.doSamplingandBuildFreqArray(inputStream , fileParti-
   tions,SamplingRatioPerPartition)
3:     if inputStream.hasGenomeFileheader then
4:         outputStream.write(GenomeFileheader)
5:     end if
6:     Proposed.writeGenomeSymbolsEncoder(outputStream)
7:     while !inputStream.EOF do
8:         genomeChar ← inputStream.GetChar().
9:         code ← Proposed.encode(genomeChar).
10:        oneByteStore.store(code).
11:        if oneByteStore.ISFull() then
12:            outputStream.write(oneByteStore).
13:            oneByteStore.empty().
14:        end if
15:    end while
16:    if !oneByteStore.ISEmpty() then
17:        outputStream.write(oneByteStore).
18:        outputStream.write(NumOfExtraBits).
19:    end if
20: end procedure
1: function DOSAMPLINGANDBUILDFREQARRAY(INPUTSTREAM , FILEPARTI-
   TIONS,SAMPLINGRATIOPERPARTITION)
2:     SamplingSize ← SamplingTimeSlots / PartitionSizes().
3:     SamplesPerPartition ← PartitionSize * SamplingRatioPerPartition.
4:     SymbolsFrequencyArray ← Integer Array with Four elements filled with ones.
5:     while filePartitions >0 do
6:         Offest ← Random.GetDouble * PartitionSize .    ▷ Random.GetDouble generates
   random numbers between 0 and 1
7:         inputStream.Seek(offest).
8:         inputStream.Read(SamplesPerPartition, dataBuffer).
9:         UpdateFrequency(SymbolsFrequencyArray, ← dataBuffer).
10:        filePartitions ← filePartitions - 1.
11:    end while
12:    FreqArray.Build(SymbolsFrequencyArray). return FreqTable
13: end function
```

4.2.2 Problem Formulation

4.2.2 The Random Sampling of Fourier Transform

We are using the random sampling of Fourier transform theory to update the content-encoding tree for big genomic symbols. Accordingly, we need to find the best representation of Fourier i$_{best}$ of X complex exponential terms to approximate i for a given a moment i $\in \mathbb{C}^N$. The fast Fourier transform (FFT) can be applied to find the i within X largest terms. Fourier representation i* can be found by sampling a subset $\tau \subseteq [0, N-1]$ of i according to [106] as follows:

$$\|i - i\|_2^2 \leq (1+\epsilon)\|i - i_{best}\|_2^2 \tag{4.1}$$

the ϵ refers to the error bound. Applying independent Bernoulli trials on the set $[0, N$ - 1] with a fixed probability to set the τ

We can find the amount of the sampling rate by employing i in the [107] as follows:

$$i[n] = \frac{1}{\sqrt{N}} \sum_{x=0}^{X-1} \alpha_{x e^{j2\pi\omega_x n/N}}, w_x \subseteq [0, N-1]. \tag{4.2}$$

That can be written in an array form such as i = $\mathbf{F}\alpha$, for the discrete Fourier transform (DFT) array $\mathbf{F}$, the elements of the DFT can be found using the formula $\mathbf{F}_{w,t} = \frac{1}{\sqrt{N}} e^{j2\pi\omega t/N}, \omega, t = 0, ..., N-1$, and α only X non-zero values in the regular repetitions ω_x. The goal is to recover α from the random samples of i. The sampling pool generator τ is the Fourier random sampling theory. The generated elements M of the sampling pool τ. Therefore, we can obtain each sample cycle through the M/N or discarded with probability $1 - M/N$. The sparse vector α of time slots can be conducting with high probability if $M = \mathcal{O}(X log^4 N)$ [108] that utilizes geometric probability distributions. This chapter applies the Fourier random sampling to generate updated encoding tree of genomic datasets to minimize the data size during transfer phase.

4.2.2 Continuous Time Random Sampling

Continue randomize of sampling intervals can be found using the following theories [109] [110] [111] [112] [113] . Then the random interval sampling $s(t)$ is defined as

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - t_n). \tag{4.3}$$

The random operation $i(t)$ can be sampled using $s(t)$ can be written as $y(t) = i(t)s(t)$. If t_n is independent from $i(t)$, then

$$\Phi_y(f) = \Phi_i(f) * \Phi_s(f), \tag{4.4}$$

where $\Phi_y(f), \Phi_i(f)$, and $\Phi_s(f)$ are the power spectral densities (PSD) of $y(t), i(t)$ and $s(t)$, respectively. When $t_n = nT$,

$$\Phi_s(f) = \frac{1}{T^2} \sum_{n=-\infty}^{\infty} \delta(f - \frac{n}{T}). \tag{4.5}$$

Therefore, the periodic sampling of the encoding tree of the genomics alphabet $\Phi_s(f)$ can use [110] to obtain the following theorem.

Theorem 2.1

Let's assume the β is the average of the sampling cycles, we can find $1/E[\tau_x]$, where τ_x refers to the independently and identically distributed (i.i.d) intervals between samples. If the characteristic function of τ_x is $\psi_{\tau_x}(f)$, then

$$\Phi_s(f) = \beta \Re \left\{ \frac{1 + \psi_{\tau_x}(2\pi f)}{1 - \psi_{\tau_x}(2\pi f)} \right\}. \tag{4.6}$$

Since $\psi_{\tau_x}(0) = 1$, $Phi_s(f)$ will have value $f = 0$.

4.2.2 Collective Random Sampling

The sampling time t_x can be found using the collective random sampling (CRS) in the following equation

$$t_x = t_{x-1} + \tau_x, \tag{4.7}$$

applying the exponential distribution $\sim$ Exp(λ) to the i.i.d interval τ_x, we can get the following equation

$$\Phi_s(f) = \lambda^2 \delta(f) + \lambda. \tag{4.8}$$

When τ_x is uniformly distributed $\tau_x \sim$ Uniform[a, b],

$$\Phi_s(f) = \begin{cases} P(\rho \frac{\sin((b-a)\pi f)}{(b-a)\pi f}), (b-a)\pi f & f \neq 0 \\ (\frac{2}{a+b})^2 \delta(f) & f = 0 \end{cases}, \tag{4.9}$$

where $P(r, \theta)$ is a Poisson kernel defined as:

$$P(r, \theta) = \frac{1 - r^2}{1 - 2r\cos\theta + r^2}, \tag{4.10}$$

4.2.2 Continuous Sampling Intervals

Implementing periodically sampling is not a minor challenge because the sampling intervals are usually divided into fixed time slots determined mostly by the CPU clock speed. Therefore, we can use rounding up Δ to specify the sampling intervals τ_x^q such that:

$$\tau_x^q = n\Delta, if (n-1)\Delta < \tau_x \leq n\Delta, \quad n \in \Omega, \tag{4.11}$$

where Ω refers to the adequate integers that can be used. For instance, we can get the Ω = 1, 2,... when the the exponential distribution is used to the τ_x^q. Whereas, the $\Omega = \{1, 2, ..., \lfloor \frac{T}{\Delta} \rfloor\}$ in case of using the uniform distribution in $[0, T]$ to the τ_x^q. The time slots of the sampling of τ_x can be found via their characteristic function $\psi_{\tau_x}(f)$. Accordingly, we ensure that the $\Phi_s(f)$ becomes periodic synchronizing to the periodicity of the $\frac{1}{\Delta}$. Consequently, we can ensure getting the best possible time slots of sampling rates only if $i(t)$ place within $[-\frac{1}{2\Delta}, \frac{1}{2\Delta}]$ limits. In other words, we can obtain the best possible encoding tree with minimum sampling intervals that includes the highest frequency of genomics symbols.

4.3 Experiments and Results

In this section, we discuss the size and transfer time of the simulated big genomic datasets by using the proposed and standard data-encoding schemes over HTTP. The examined datasets in both formats, FASTA and FASTQ, were generated via our simulated NGS generator, as can been seen in Tables 4.2 on page 87 and 4.3 on page 88.

4.3.1 Experimental Setup

This chapter applies the proposed and standard data-encoding schemes during transfer of big genomic datasets using HTTP. Several sizes of genomic datasets in both formats, FASTA and FASTQ, were generated using our NGS simulation and tested up to 1TB to validate our data-encoding. The experiments were performed on machines that have specifications shown in Table 4.1 on page 86.

4.3.2 Experimental Results

Our experimental results of big genomic datasets are shown in Tables 4.2 on page 87, 4.3 on page 88, 4.4 on page 89, and 4.5 on page 89, and Figures 4.3 on page 87, and 4.4 on page 88.

In order to assess the effectiveness of our proposed data minimization method, several genomic dataset sizes were tested using both data-encoding methods: proposed and standard during transfer using HTTP. The results show that the proposed method significantly decreases the dataset sizes, and then shortens the transfer time, as shown in Tables and Figures listed above.

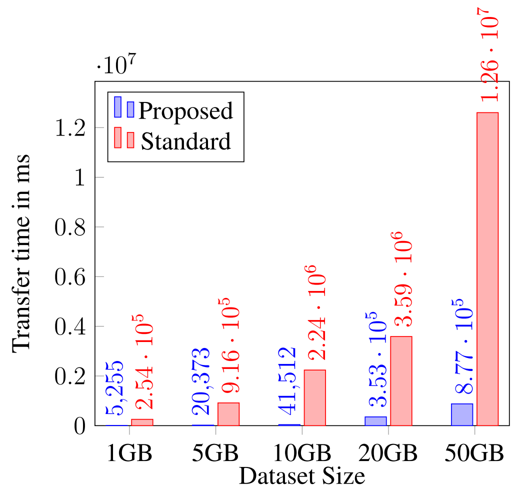
Table 4.1: Experimental setup

Specifications	Details
Processor	2.4 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 4000 1024 MB
Operating System	Windows 8.1 Pro
Download	87 Mb/s
Upload	40 Mb/s
Programming Language	C# .Net
Used Protocol	HTTP
Dataset Sizes	1GB,5GB,10GB,20GB,50GB,100GB,200GB,400GB,500GB,1TB

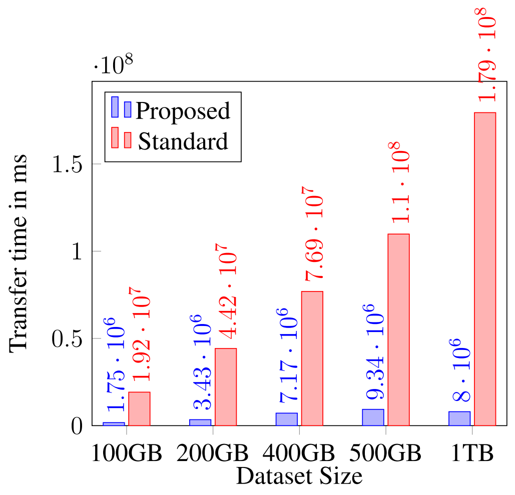
For example, the original 1GB dataset in FASTA format was minimized to 293,393 KB using the proposed method during transfer phase in contrast to 1,048,576 KB using the standard encoding method with about 72% size reduction. Consequently, the transfer time was 5,255 milliseconds (ms) using the proposed method, while 2.54 x $10^5$1 ms resulted by using the standard scheme with about 98-fold of time acceleration. Also, 223,136,461 KB were transferred of 1TB FASTA dataset using the proposed method, while 1,073,741,824 KB resulted by using the standard method for the same dataset with about 79% size reduction. The transfer time of 1TB FASTA dataset using the proposed data-encoding were 8 x 10^6 ms, while 1.79 x 10^8 ms resulted in the transfer of the same dataset using the standard scheme, with about 96-fold of time acceleration. Also, the 1GB FASTQ dataset was minimized to 356,516 KB by using the proposed scheme, while 1,048,576 KB resulted by using the standard scheme, with about 34% of size reduction. Consequently, 8,333 ms were needed to transfer the dataset by using the proposed method, while 15,150 ms were required to transfer the same dataset using the standard approach, with 55-fold of time acceleration. Moreover, 1TB FASTQ dataset was minimized to 418,759,311 KB by using a proposed encoding method, while 1,073,741,824 KB transferred using the standard approach with about 39% of size reduction. The time acceleration of the 1TB FASTQ dataset was 54-fold by sending the dataset in 7.74 x 10^6 ms using the proposed method, while 143 x 10^7 ms were needed to transfer the same dataset. The maximum size reduction of examined FASTA datasets using the proposed data-encoding scheme is 79% and about 98-fold of time acceleration, The maximum rate of size reduction for the examined FASTQ datasets using the proposed encoding method is 45%, while 57-fold is the maximum time acceleration, as shown in Figure 4.5 on page 90.

Table 4.2: Simulated next-generation sequencing (NGS) dataset sizes, FASTA format

Datasets	**Data-Encoding Size in KB**		**Number of Genomic Symbols in each Dataset**			
	Proposed	**Standard**	**A**	**C**	**G**	**T**
1GB	293,393	1,048,576	536,870,912	214,748,160	214,748,160	107,374,592
5GB	1,181,950	5,242,880	3,758,095,360	268,431,360	429,496,320	912,686,080
10GB	2,731,022	10,485,760	6,442,444,800	1,610,608,640	858,992,640	1,825,372,160
20GB	4,722,414	20,971,520	15,461,867,520	2,147,471,360	1,717,985,280	2,147,512,320
50GB	14,369,917	52,428,800	26,843,545,600	13,421,772,800	9,663,641,600	3,758,131,200
100GB	23,321,911	104,857,600	73,014,374,400	10,737,356,800	2,147,430,400	21,475,020,800
200GB	53,935,177	209,715,200	1.33144E+11	21,474,713,600	25,769,779,200	34,359,910,400
400GB	99,176,254	419,430,400	2.96353E+11	47,244,492,800	42,949,427,200	42,950,246,400
500GB	127,617,346	524,288,000	3.22122E+11	1.07374E+11	96,636,416,000	10,738,176,000
1TB	223,136,461	1,073,741,824	7.91648E+11	1.31941E+11	1.64926E+11	10,996,416,512

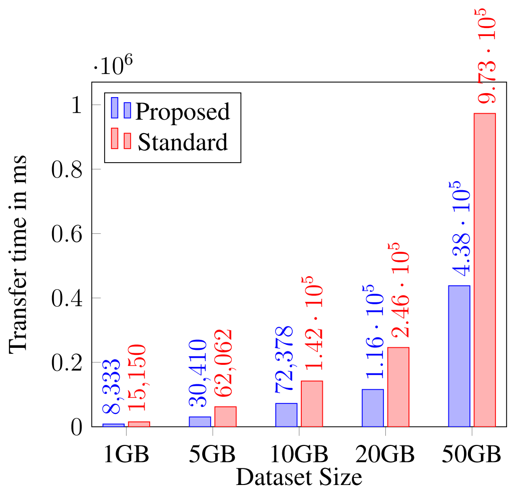


(a) Transfer time of FASTA datasets: 1G, 5GB, 10GB, 20GB, and 50GB

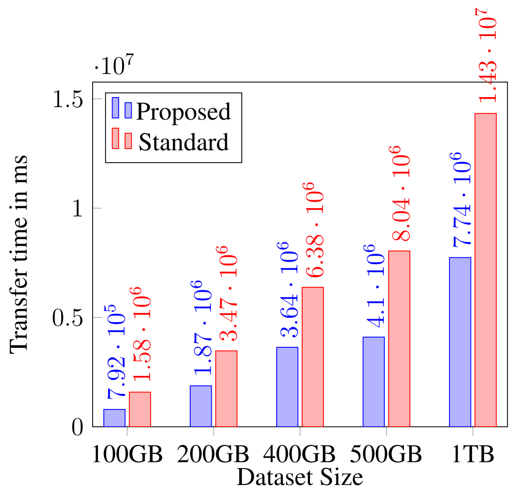


(b) Transfer time of FASTA datasets: 100G, 200GB, 400GB, 500GB, and 1TB

Figure 4.3: Transfer time of simulated FASTA datasets G, 5GB, 10GB, 20GB, 50GB, 100G, 200GB, 400GB, 500GB, and 1TB using proposed and standard data-encoding approaches



(a) Transfer time of FASTQ datasets: 1G, 5GB, 10GB, 20GB, and 50GB



(b) Transfer time of FASTQ datasets: 100G, 200GB, 400GB, 500GB, and 1TB

Figure 4.4: Transfer time of simulated FASTQ datasets G, 5GB, 10GB, 20GB, 50GB, 100G, 200GB, 400GB, 500GB, and 1TB using proposed and standard data-encoding approaches

Table 4.3: Simulated next-generation sequencing (NGS) dataset sizes, FASTQ format

Datasets	**Data-Encoding Size in KB**		**Number of Genomic Symbols in each Dataset**			
	Proposed	**Standard**	**A**	**C**	**G**	**T**
1GB	356,516	1,048,576	1,100,243	1,171,475	1,194,506	1,070,103
5GB	2,097,152	5,242,880	4,587,490	4,722,908	4,853,198	4,293,654
10GB	3,879,731	10,485,760	10,579,317	10,871,037	11,069,642	10,603,326
20GB	8,808,038	20,971,520	18,383,332	19,099,195	19,068,835	16,777,600
50GB	20,971,520	52,428,800	55,189,255	58,677,592	55,350,383	5,5861,160
100GB	33,554,432	104,857,600	90,769,611	98,405,480	100,395,898	93,309,194
200GB	79,691,776	209,715,200	202,744,248	208,871,811	217,755,796	209,397,412
400GB	188,743,680	419,430,400	387,005,677	391,542,999	392,718,494	375,820,583
500G	214,958,080	524,288,000	490,035,888	508,771,266	550,501,004	49,066,4651
1TB	418,759,311	1,073,741,824	864,354,300	910,476,782	892,603,677	768,828,857

4.4 Conclusions

In this chapter, we implemented an innovative data minimization algorithm that relies on a combination of the naive bit encoding approach and the random sampling of Fourier transform theory to form a time-based random sampling data-encoding. The proposed algorithm is designed

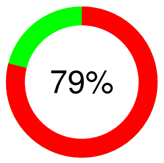
Table 4.4: Size reduction and time acceleration of datasets, FASTA format

Dataset	Size Reduction	Time Acceleration
	Proposed vs. Standard	Proposed vs. Standard
1GB	72%	98x
5GB	77%	98x
10GB	74%	98x
20GB	77%	90x
50GB	73%	93x
100GB	78%	91x
200GB	74%	92x
400GB	76%	91x
500GB	76%	91x
1TB	79%	96x

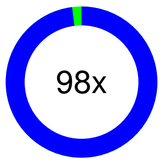
Table 4.5: Size reduction and time acceleration of datasets, FASTQ format

Dataset	Size Reduction	Time Acceleration
	Proposed vs. Standard	Proposed vs. Standard
1GB	34%	55x
5GB	40%	49x
10GB	37%	51x
20GB	42%	47x
50GB	40%	45x
100GB	32%	50x
200GB	38%	54x
400GB	45%	57x
500GB	41%	51x
1TB	39%	54x

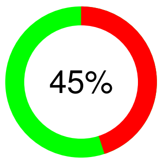
to be integrated with transfer protocols to reduce the size of the real-time big genomic datasets in both formats: FASTA and FASTQ, during the transfer phase, and then to transfer the data securely in a shorter time. The results indicate that the proposed data minimization algorithm is capable or reducing the transfer time up to 98-fold of FASTA datasets and 57-fold of FASTQ datasets,



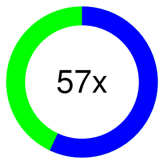
(a) Size reduction of FASTA



(b) Time acceleration of FASTA



(c) Size reduction of FASTQ



(d) Time acceleration of FASTQ

🟢 Proposed (time/size) 🔴 Standard (size) 🔵 Standard (time)

Figure 4.5: The maximum size reduction and time acceleration of tested genomics datasets

compared to the standard data-encoding on tested datasets. Further, the results also show the dataset can be reduced up to 79% of the original FASTA format dataset sizes and up to 45% of FASTQ format.

The proposed data minimization approach is integrated to HTTP to assess how the transfer protocol behaves in terms of transfer time and size. Also, we showed that our data minimization algorithm provides a significant size reduction and transfer time acceleration, as well as adds an extra level of security during the transfer because dataset symbols encode in different codewords during the transfer phase. We demonstrated that the data size can be significantly reduced by our hybrid data-encoding, compared to a standard scheme. Also, we implemented an NGS simulator to generate genomic datasets in both formats: FASTA and FASTQ, to verify the performance of our current data minimization scheme and compared it to the standard. Our experiments indicated that the Fourier transform theory-based random sampling, data-encoding performs much better than the standard scheme by ensuring the assignment of short codewords for the genomic dataset symbols. We conclude that the proposed data minimization algorithm provides the best performance among current data-encoding approaches for big real-time generated genomic datasets.

CHAPTER 5

VARIABLE AND NAIVE BIT-BASED DATA MINIMIZATION ALGORITHMS FOR FAST AND SECURE TRANSFER OF BIG GENOMIC DATASETS

5.1 Introduction

Modern genomic studies utilize high-throughput instruments which can produce data at an astonishing rate. These big genomic datasets are produced by using next generation sequencing (NGS) machines that can easily reach a peta-scale level creating storage, analytic and transmission problems for large-scale system biology studies. Traditional networking protocols are oblivious to the data that is being transmitted and are designed for general purpose data transfer. In this chapter, we present a novel data-aware network transfer protocol to efficiently transfer big genomic data. Our protocol exploits the limited alphabet of the DNA nucleotide and is developed over the hypertext transfer protocol (HTTP) framework. Our results show that the proposed technique improves transmission up to 84 times when compared to normal HTTP encoding schemes. We also show that the performance of the resultant techniques, using a single machine, is comparable to BitTorrent protocol used on 10 machines.

Next generation sequencing (NGS) machines, such as the Illumina HiSeq2500, can generate up to 1TB of data per run, and the data grow exponentially for large systems biology studies [114]. More often than not, these large genomic data sets have to be shared with fellow scientists or with cloud services for data analysis. The usual practice is to transfer the data using a networking protocol such as HTTP or FTP. Traditional networking protocols are oblivious to the data that is being transmitted and are designed for general purpose data transfer. Consequently, transfer is typified by an exceedingly long time when large data sets are involved. Previous methods to improve transmission have focused on using FTP/HTTP protocols and multiple machines to increase throughput [115]. However, those solutions are inefficient in terms of hardware and do not exploit the additional data redundancy of DNA sequences for efficient transmission.

This chapter introduces two data minimization techniques to reduce big genomic datasets during data transfer processes that are variable-length and naive bit encoding schemes. We assert that if the scope of the data is known, such as genomic data, then networking protocols should be able to take advantage of this information for better efficiency. The key idea of VTTP is utilizing variable length codewords for DNA nucleotides in the content-encode of HTTP to maximize network resources usage [19]. Our proposed transfer techniques decreases the size of the data that need to be transmitted via assigning the shortest possible codewords for repeated symbols, hence shortening the transfer time. The proposed data encoding schemes do not require any extra hardware resources and is shown to be much faster than other competing techniques and protocols.

5.1.1 Contribution

Creating the proposed content encoding mechanism relies on assigning variable-length binary codewords for the genomic symbols based on the frequency of the nucleotides in the dataset. The VTTP dynamically switches between the traditional charsets for symbols that do not belong to the genomic charset (A, G, C, T) and to our efficient encoding for DNA nucleotides. Lengths of genomic charset codewords is a static variable-length in range of 1-3 bits long. We have implemented our encoding technique on top of HTTP for its universality and flexibility on various platforms. We are not aware of any other data-aware protocols that exploit redundancy in genomic data for efficient transmission. This VTTP is an improvement over our earlier work that used fixed-length codewords for genomic symbols i.e. 2-bit long for each character [79].

5.1.2 Chapter Goals and Organization

The goal of this chapter is the design and implementation of a data-aware transfer protocol that relies on HTTP using variable and naive bit methods. We implement the data minimization techniques by modifying the HTTP content encoding approach to transfer a big genomic dataset. We compare our results with traditional HTTP, FTP and BitTorrent-like transfer protocols to transfer large genomic datasets.

The chapter is organized as follows: Section 5.2 discusses the overall architecture of the proposed variable-length-based content-encoding algorithm model description, formulation, and results. The second architecture of the proposed naive-bit-based content-encoding algorithm model description, formulation, and results are discussed in Section 5.3. Finally, we discuss future work and our conclusions in Section 5.4.

5.2 Proposed Variable-Length-Based Model of Content-Encoding Technique

In this section, we illustrate our implementation of VTTP that utilizes VLBE for content encoding. Model formulations are also discussed for different possible scenarios of symbol repetitions to compare our proposed encoding to the current method used in HTTP encoding:

5.2.1 Model Description

This subsection presents our implementation of HTTP that relies on VLBE content encoding to transfer big genomic datasets. This model assigns short variable codewords for genomic symbols. The fact that a genomic alphabet consists of only 4 symbols A, G, C, T makes it an ideal candidate for VLBE encoding and can be represented in less than 8-bits. We can encode the genomic dataset symbols in 4 unique decipherable codewords i.e. [0, 11, 100, 101] or simply [0, 3, 4, 5] as shown in Figure 5.1 on page 95. HTTP, in most browsers, starts when the client searches for specific data, and the HTTP client side initiates a connection with the server that contains the required data. The connection between the client and the server establishes a 3-way handshake using the TCP/IP protocol. After establishing the connection, the client sends a request for certain dataset(s) to the server that checks the header(s), the method(s), and the resource address(es). The server retrieves the required data and starts to convert file symbols to binary form by using a VLBE and passes it to a compression technique (GZIP in this implementation). FASTA file for a single sequence is described by a title line followed by one or more data lines. The title line begins with a right-angle bracket followed by a label. The label ends with the first white space character. The data lines begin right after the title line and contain the sequence characters in order as shown in Figure 5.1 on page 95. At this point, we read the first line of the FASTA file [87] using an ASCII character set, and the remaining lines are read using VTTP. The server starts encoding using the VLBE character set, compresses via GZIP, and transfers the data (response) through the network medium. The compressed data are received by the client, along with header(s) and method(s) to store it. The client starts to decompress the received data using GZIP to obtain the binary form.

We utilize a binary tree as a structure to represent our VLBE because it is faster to search, avoids duplicate values, and is easy to decode at a receiver side. Assuming we have a file of 18 symbols with different symbol repetitions, the following: a_1 appears in a frequency rate of 61%; a_2 has a repetition rate of 17%, and 11% for both a_3 and a_4 as shown in Figure 5.1 on page 95. In this example, the file has redundancy and VLBE works by assigning a variety of bit lengths reaching 1 bit per symbol (bps). There are 3 possible code lengths for the symbol a_1 in this example, as appears in Figure 5.1 on page 95. The proposed method produces better results in contrast to the FLBE.

Therefore, encoding 18 symbols in 29 bits yields an average of 1.6 bits/symbol in VLBE, as compared to 144 bits in the current HTTP encoding. The 3 VLBE possibilities of this example show 3 different code lengths (29, 37 and 47) bit long. However, VLBE still assigns short codes for the whole string, in contrast to FLBE for the real-time applications i.e. data transfer.

We designed variable codes in Figure 5.1 on page 95 in a way that makes it easy to decode using a prefix property (unambiguously). This property assigns a unique binary codeword for each single alphabet symbols to facilitate the decode operation on the client side. The variable length binary encoding has been used for static applications, but transferring dynamic decoding trees has not been investigated.

It is important to note that the proposed VLBE does not guarantee minimal codewords for the data since the frequency of the data is not calculated due to the complexity of the computation process for big data. However, it is expected that the proposed strategy will transfer data much more rapidly as compared to the traditional HTTP protocol. For the current implementation,

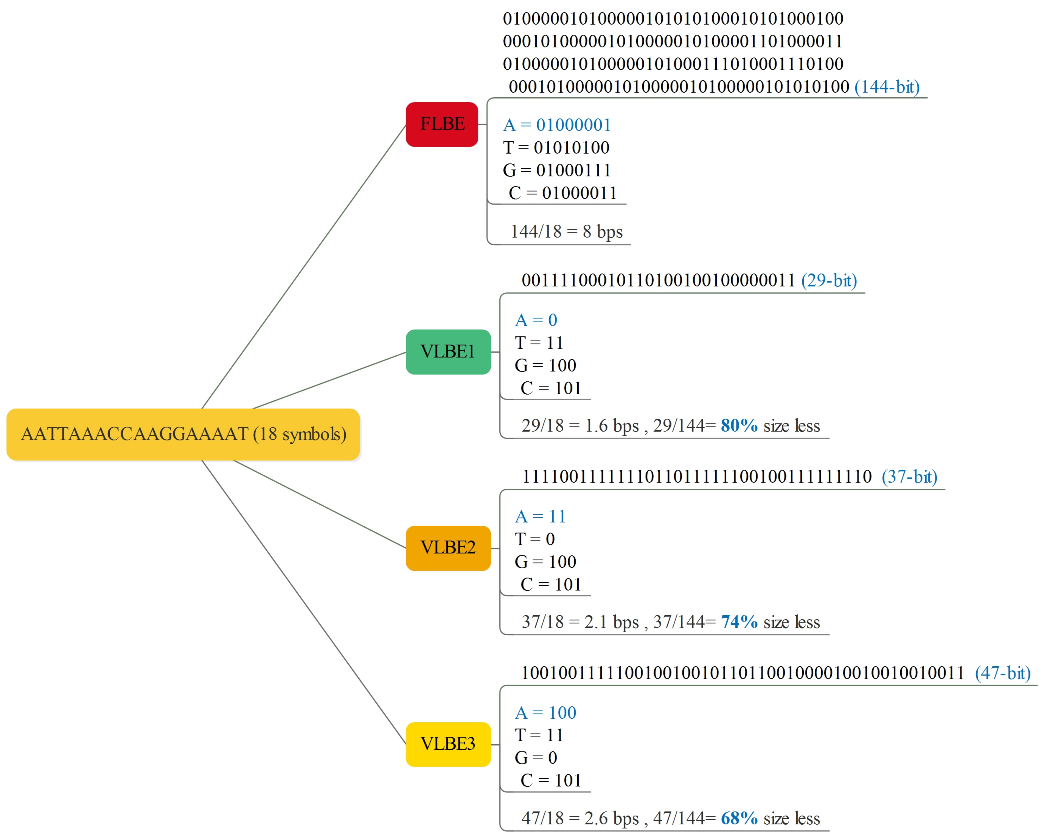

AATTAAACCAAGGAAAAT (18 symbols)
FLBE
010000010100000101010100010101000100
0001010000010100000101000011010000 11
010000010100000101000111010001110100
000101000001010000010100000101010100 (144-bit)
A = 01000001
T = 01010100
G = 01000111
C = 01000011
144/18 = 8 bps
VLBE1
00111100010110100100100000011 (29-bit)
A = 0
T = 11
G = 100
C = 101
29/18 = 1.6 bps , 29/144= 80% size less
VLBE2
1111001111111011011111100100111111110 (37-bit)
A = 11
T = 0
G = 100
C = 101
37/18 = 2.1 bps , 37/144= 74% size less
VLBE3
10010011111001001001011011001000010010010010011 (47-bit)
A = 100
T = 11
G = 0
C = 101
47/18 = 2.6 bps , 47/144= 68% size less

Figure 5.1: Example of the HTTP content encoding schemes for a string of 18 symbols with variety of repetitions. 3 possible codes for the genomic symbols [A, T, G and C]

the following are required: a 1-bit codeword length for a genomic symbol that has the highest occurrence (based on a local sample); and a 2-bit codeword length for a symbol with next largest repetition; and 3-bit codeword length for the remaining two symbols. The pseudocode for our protocol can be seen in Algorithm 5 on page 96.

Algorithm 5 VLBE-based HTTP

```
1: procedure ENCODING
2:     if !inputStream.hasGenomicFileheader then
3:         STOP (IS NOT Genomic File)
4:     else
5:         Encode the whole first line using a traditionalChar
6:     end if
7:     VLBE.writeGenomicSymbolsEncoder(outputStream)
8:     while !inputStream.EOF do
9:         if inputStream.GetChar() $\in \{A, T, G, C\}$ then
10:            genomicChar $\leftarrow$ inputStream.GetChar()
11:            code $\leftarrow$ VLBE.encode(genomicChar)
12:        else
13:            traditionalChar $\leftarrow$ inputStream.GetChar()
14:            code $\leftarrow$ VLBE.encode(traditionalChar)
15:        end if
16:        oneByteStore.store(code)
17:        if oneByteStore.ISFull() then
18:            outputStream.write(oneByteStore)
19:            oneByteStore.empty()
20:        end if
21:    end while
22:    if !oneByteStore.ISEmpty() then
23:        outputStream.write(oneByteStore)
24:        outputStream.write(NumOfExtraBits)
25:    end if
26: end procedure
```

5.2.2 Problem Formulation

Our analysis starts from the fact that in practical cases, the transfer time of data fluctuates due to several reasons such as bandwidth and message loss. The data transfer throughput (*Th*) is measured by the minimum time needed (*t*) to transfer certain data amount (*N*) from the sender to the receiver. In order to minimize transfer time, we need to either maximize the bandwidth (which costs more) or minimize the data size, which will reduce overall resources and time and is being pursued in this work. The transfer throughput can be formalized in Equations 5.1, 5.2 on page 97:

$$Th = \frac{N}{t} \tag{5.1}$$

A higher *Th* means better protocol throughput via transferring a large data amount in less time. Consequently, the protocol throughput increases when a bit per symbol (bps) is reduced as much as possible. For example, transferring *N* string symbols in *B* bits indicates efficiency of the encoding scheme as shown in the following:

$$bps = \frac{B}{N} \tag{5.2}$$

Here the minimum *bps* is better, and hence, shows a more time-efficient performance as compared to the original 8-bit transfers. The VLBE scheme utilizes all unused space in each single byte, a fact that reduces the transfer time by decreasing data size in the next phases (compression, transfer, decompression, decoding to plain text). To simplify our model, let's assume we have an A alphabet as $\{a_1, a_2, a_3, ..., a_n\}$ that consists of *n* symbols, for the genomic dataset, A $\in$ {A, T, G, C}(nucleotides). The codeword can be represented in C(A) $\in$ {0,11,100,101}. The *time* complexity for encoding a string of *N* symbols using the HTTP fixed encoding is O(N) and the *space* complexity *S* can be calculated in Equation 5.3 on page 97:

$$S_{http} = \sum_{i=1}^{n} a_i * 8 \tag{5.3}$$

The space complexity of our proposed encoding scheme can be formulated in Equation 5.4 on page 97:

$$S_{vlbe}(A) = \sum_{i=1}^{n} a_i * codeword_{length} \tag{5.4}$$

VTTP has a time complexity of O(N/P), where P depends on the connection bandwidth and *bps*. Also, we can divide and formulate our model costs (C) into the following Equations:

$$C_{total} = C_{computation} + C_{communication} \tag{5.5}$$

$$C_{computation} = O\left(C_{header\ check} + C_{encoding} + C_{compression}\right) \tag{5.6}$$

$$C_{communication} = O\left(C_{3\ way\ handshake} + C_{bandwidth}\right) \tag{5.7}$$

The Equations 5.5, 5.6, 5.7 on page 98 show that the computation of VTTP consumes additional time to encode symbols since it switches between two charsets during the reading of the file. However, it takes much less time in the next steps to shorten the transfer time many times.

5.2.3 Experiments and Results

In this section, we discuss FTP and HTTP behaviors using both the current (fixed) and the proposed (variable) length encoding schemes for a variety of genomic datasets. The examined datasets that are FASTA format files were downloaded through two sources: National Center for Biotechnology Information (NCBI) [89] and University of California Santa Cruz (UCSC) [90] websites as shown in Table 5.1 on page 99.

5.2.4 Experimental Setup

This chapter compares the proposed VTTP with FTP, HTTP and BitTorrent-like transfers. Several datasets (up to 430GB of FASTA files) have been fed to these implementations to validate our approach. The experiments were performed on machines that have specifications shown in Table 5.2 on page 99.

Table 5.1: Experimental datasets

IDs	Source	Size(KB)	Renamed
pataa	NCBI	563,318	1
refGeneexonNuc	UCSC	639,183	2
envnr	NCBI	1,952,531	3
hg38	UCSC	11,135,899	4
patnt	NCBI	14,807,918	5
gss	NCBI	30,526,525	6
estothers	NCBI	43,632,488	7
humangenomic	NCBI	45,323,884	8
othergenomic	NCBI	346,387,292	9

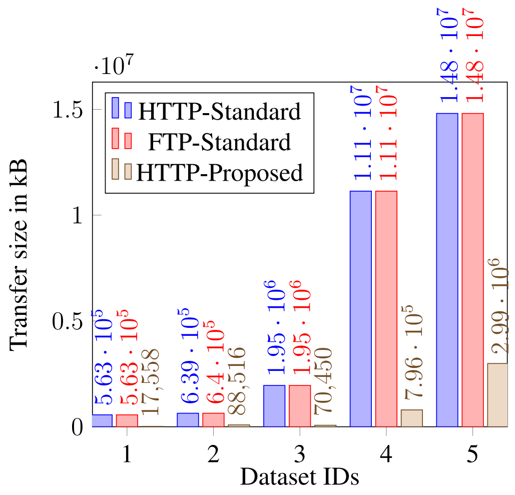
Table 5.2: Experimental setup

Specifications	Details
Processor	2.4 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 4000 1024 MB
Operating system	Windows 8.1 Pro
Download	87 Mb/s
Upload	40 Mb/s
Programming Language	C# .Net
Protocols	FTP, HTTP, BitTorrent and VTTP
Dataset sizes	550MB - 340GB

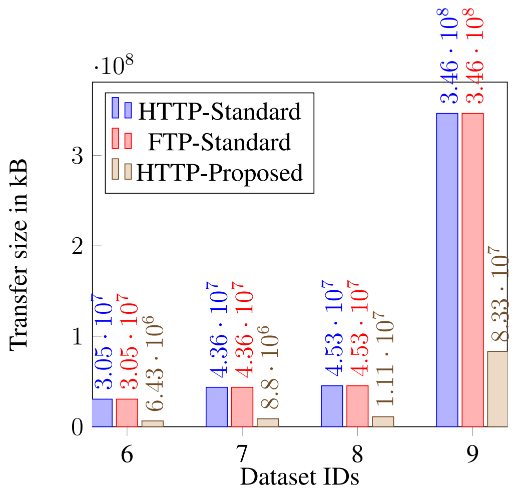
5.2.5 Experimental Results

Our experimental results as shown in Figures 5.2 on page 100, 5.3 on page 100, 5.8 on page 108, Tables 5.3 on page 102, 5.4 on page 102, 5.5 on page 103, and 5.6 on page 103.

As can be observed, VLBE decreases the size of the data that needs to be transferred sharply and the corresponding decrease in the transfer time occurs rapidly. As can be seen in Tables Tables 5.3 on page 102, 5.4 on page 102, 5.5 on page 103, and 5.6 on page 103, 1.20 x 10^5 milliseconds (ms) are required to transfer 550MB dataset using the traditional HTTP encoding,

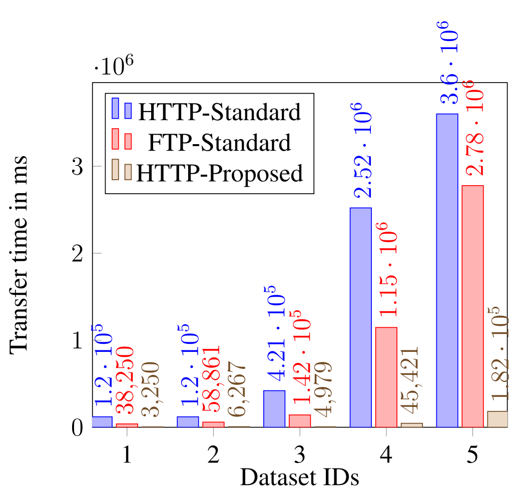


(a) Transfer size of FASTA datasets: 1 - 5

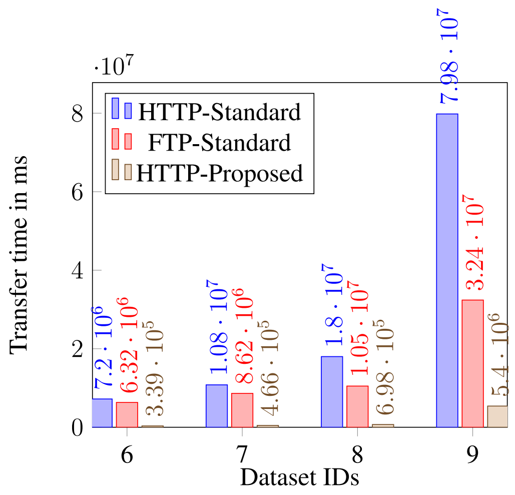


(b) Transfer size of FASTA datasets: 6 - 9

Figure 5.2: Transfer size of real FASTA datasets 1 - 9 using proposed and standard data-encoding approaches over HTTP and FTP



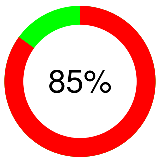
(a) Transfer time of FASTA datasets: 1 - 5



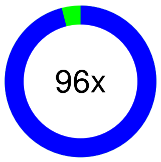
(b) Transfer time of FASTA datasets: 6 - 9

Figure 5.3: Transfer time of real FASTA datasets 1 - 9 using proposed and standard data-encoding approaches over HTTP and FTP

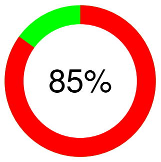
3.82×10^4 ms via the FTP whereas 3.25×10^3 ms are required to transfer the same file via the HTTP-VLBE. This rate of transfer is approximately 37 times faster than HTTP-FLBE and about 12 times faster than FTP. Also, the 30GB dataset was transferred in 7.20×10^6 ms using the HTTP, 6.31×10^6 ms by FTP, whereas it took only 3.39×10^5 ms to transfer the file using VLBE. This is approximately 21 times faster than HTTP and 18 times faster than FTP. We show results for up to



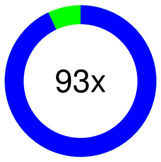
(a) Average of size reduction using proposed encoding scheme compares to standard one over HTTP



(b) Average of time acceleration using proposed encoding scheme compares to standard one over HTTP



(c) Average of size reduction using proposed encoding scheme compares to standard one over FTP



(d) Average of time acceleration using proposed encoding scheme compares to standard one over FTP

🟩 Proposed (time/size) 🟥 Standard (size) 🟦 Standard (time)

Figure 5.4: The maximum size reduction and time acceleration of tested genomics datasets

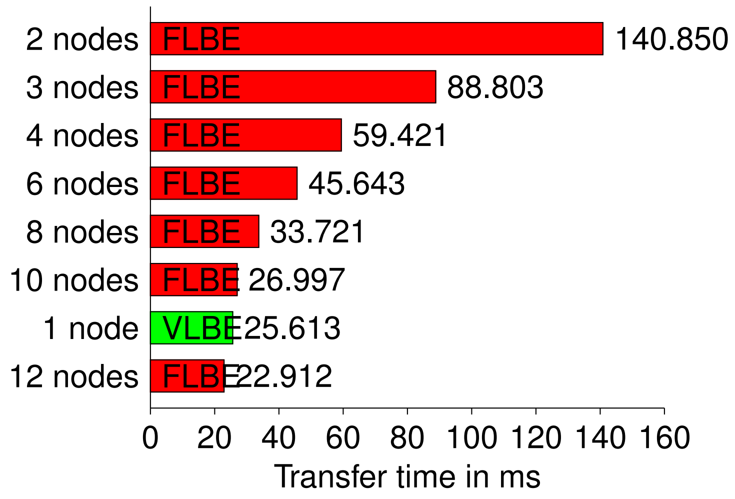


Figure 5.5: Transfer time of 1G FASTA dataset using (1) VLBE-based machine and up to (12) FLBE-based machines work in parallel

Table 5.3: Comparison between genomic dataset sizes in (kB) using standard and proposed data-encoding approaches over HTTP and FTP

Dataset IDs	Standard		Proposed
	HTTP	FTP	HTTP
1	5.63+05	5.63+05	1.75+04
2	6.39+05	6.39+05	8.85+04
3	1.95+06	1.95+06	7.04+04
4	1.11+07	1.11+07	7.95+05
5	1.48+07	1.48+07	2.98+06
6	3.05+07	3.05+07	6.42+06
7	4.36+07	4.36+07	8.79+06
8	4.53+07	4.53+07	1.10+07
9	3.46+08	3.46+08	8.32+07

Table 5.4: Comparison between genomic dataset transfer time in (ms) using standard and proposed data-encoding approaches over HTTP and FTP

Dataset IDs	Standard		Proposed
	HTTP	FTP	HTTP
1	1.20+05	3.82+04	3.25+03
2	1.22+05	5.88+04	6.26+03
3	4.20+05	1.41+05	4.97+03
4	2.52+06	1.14+06	4.54+04
5	3.60+06	2.77+06	1.82+05
6	7.20+06	6.31+06	3.39+05
7	1.08+07	8.62+06	4.65+05
8	1.80+07	1.04+07	6.97+05
9	7.98+07	3.24+07	5.39+06

340GB. The average decrease in the size of the data sets as compared to HTTP and FTP is around 15 times.

The corresponding decrease in the running time is 33 times faster compared to HTTP, and 16 times faster compared to FTP over all data sets. In order to compare the results of the proposed approach with that of existing BitTorrent protocol, we implemented the latter approach as

Table 5.5: Comparison between tested genomic dataset size using proposed and standard data-encoding schemes

Dataset IDs	**Dataset Size Reductions**		
	Proposed vs. Standard-HTTP	Proposed vs. Standard-FTP	HTTP vs FTP
1	97%	97%	0%
2	88%	88%	0%
3	95%	95%	0%
4	93%	93%	0%
5	80%	80%	0%
6	79%	79%	0%
7	80%	80%	0%
8	75%	75%	0%
9	76%	76%	0%

Table 5.6: Comparison between tested genomic datasets transfer time using proposed and standard data-encoding schemes

Dataset IDs	**Dataset Time Acceleration**		
	Proposed vs. Standard-HTTP	Proposed vs. Standard-FTP	HTTP vs FTP
1	98x	94x	68x
2	96x	91x	51x
3	99x	96x	66x
4	98x	96x	55x
5	96x	94x	23x
6	95x	95x	12x
7	96x	95x	20x
8	96x	93x	42x
9	93x	83x	59x

well. The results are shown in Figure 5.5 on page 101 for a 1GB FASTA file that was downloaded from the NCBI website (Homo_sapiens.GRCH38.dna_sm_toplevel). Only 1 machine (server) is used to transfer the same file using VLBE, while *n* machines are used to transfer a file utilizing HTTP over the BitTorrent protocol.

As expected, with the increasing number of machines, the time to transfer decreases sharply over BitTorrent. It can also be observed that the transfer time required for 1GB of genomic file using our proposed protocol (VTTP) with using only 1 machine is approximately equivalent to 10 machines used in parallel with BitTorrent protocols. That reduction happens due to our encoding strategy. The results are presented for 1GB file only, and the performance of VTTP is expected to increase with increasing size of the data due to increase in redundancy. Also note that employing VTTP for multiple machines will massively decrease the time needed to transfer a file of a given size.

5.3 Proposed Naive Bit-Based Model of Content-Encoding Technique

Genomic data processing applications continue to grow in their scope, ambition, and functionality. Therefore, we believe creating a data-aware transfer protocol would optimize each byte of sent data, which increases network throughput, saves bandwidth, time, and resources. This chapter proposes a new network transfer protocol that relies on HTTP with some modifications to meet genomic requirements. Our chapter assumes that the data consists of genomic data with only four base pairs (A, T, G, C) that need to be transferred, processed, visualized, and exchanged over networks. Thus, modifying the HTTP content-encoded mechanism represents the key idea of this work, since the HTTP contains many headers [116] that offer different functions for data transmission.

HTTP headers help to manage the transfer of a variety of data types, including video and audio, over TCP and UDP protocols. Content-Encoded is responsible for managing encoding and compressing algorithms, as well as specifying the character set of transferred data. The simple form of HTTP is a request-response mode after establishing a connection between client (browser) and server over TCP. The purpose of this work is to encode each of the four bytes into one byte by assigning two bits for each genomic letter instead of eight bits. The proposed encoding scheme reduces the amount of data that needs to be transferred by three fourths (3/4) since it converts every 32 bits to 8 bits.

Moreover, the proposed encoding approach decreases significantly the transfer time because fewer data need to be transferred. After the customized encoding, the encoded-data are passed into a compression algorithm, such as GZIP, to compress data [8]. This chapter will demonstrate how GTTP improves the network performance via minimizing latency and maximizing throughput. In this chapter, the performance of GTTP is evaluated in the contexts of size and transfer time by using a TCP as a baseline for performance evaluation.

A complete cycle of transportation in HTTP begins by establishing a client-server connection using 3-way TCP handshaking. After that, the client sends a request with some headers using certain supported methods. The server receives the request and checks the headers to determine the needed contents via an accepted data type and content encoding. The GTTP determines the location of the required data to encode, compress, and to transfer over the network. The main contribution begins when the server starts transferring needed data by encoding into 2-bit form instead of 8-bit. GTTP reads each of the 4 letters and stores them in only one letter to reduce data by 75%.

Generated encoded data (25% of the original) will be passed into the compress algorithm (GZIP), which is HTTP built-in to reduce data yet again by 75%. The binary encoded form for each genomic letter can be found in Table 5.7 on page 109. For instance, GTTP encoding will encode genomic ASCII of AAAA into 00000000 binary form and save 24-bit to pass 8-bit into the compression stage and that reduces data by 75%.

The total of transmitted data would be 6.25% of original data or less (25% from encoded stage * 25% from compress stage) as shown in Equations 5.8 on page 105, 5.9 on page 106, and Table 5.7 on page 109. Let assume we have genomics file *Gen* = $\{a_1, a_2, a_3, ..., a_n\}$ $a_i \in \{$A, T, G, C$\}$

Where n is the number of characters in the file and each character will encoded into $c(ai)$ as $c(ai)$ = $\{$A = 00, T = 01, G = 10, C = 11$\}$ = 2 bit From that we create a new Equation 5.8 on page 105 as follows:

$$B(Gen) = \sum_{i=1}^{n} a_i length(c_i) \tag{5.8}$$

Where *B(Gen)* is binary convert of *Gen* file, and $length(c_i)$ represents the length of coded a_i in bits. For example, suppose we want to send a FASTA file (genomics) that contains 100,000 characters. Then we would get in:

(1) Normal coding requires 100,000 * 8 = 800,000 bits, (2) Proposed coding uses only 100,000 * 2 = 200,000 bits with 75% saving. Therefore, we obtain on a new Equation 5.9 on page 106 as follows:

$$B(Gen) = \frac{1}{4}\sum_{i=1}^{n} a_i length(c_i) \tag{5.9}$$

To get 800,000 * 0.25 = 200,000 bits in normal way whereas 200,000 * 0.25 = 50,000 bits in the proposed approach. Consequently, we will end up with sending 0.0625 of original data, which is much more efficient than existing techniques.

GTTP algorithm can be summarized in Algorithm 6 on page 106.

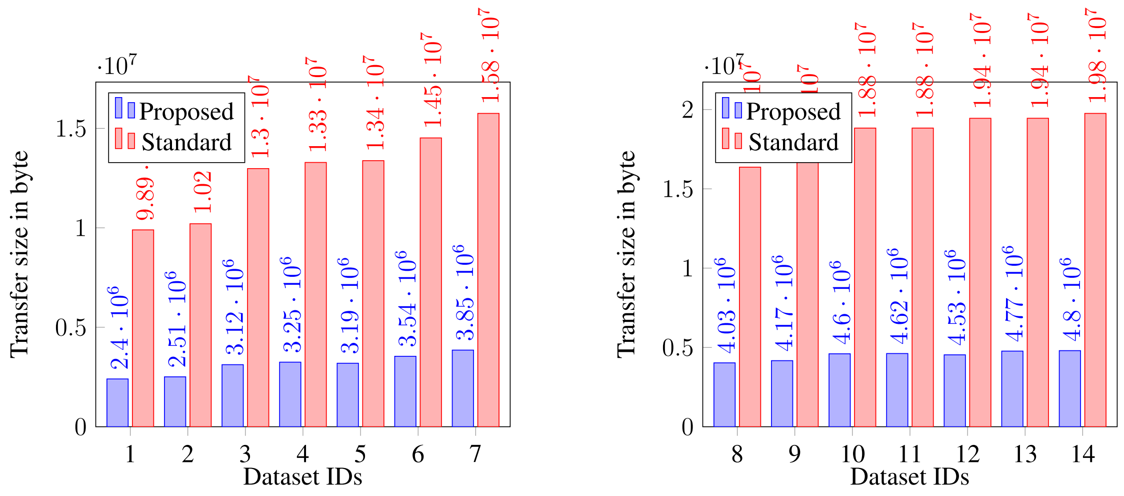
Algorithm 6 Proposed 2-bit content-encoded

```
procedure ENCODING
    if inputStream.hasGenomeFileheader then
        outputStream.write(GenomeFileheader)
    end if
    while !inputStream.EOF do
        $genomeChar \leftarrow inputStream.GetChar()$.
        $twoBits \leftarrow towBitsEncoding(genomeChar)$.
        oneByteStore.store(twoBits).
        if oneByteStore.ISFull() then
            outputStream.write(oneByteStore).
            oneByteStore.empty().
        end if
    end while
    if !oneByteStore.ISEmpty() then
        outputStream.write(oneByteStore).
        outputStream.write(NumOfExtraBits).
    end if
end procedure
```

Briefly, the first stage produces only 25% of the original data, and the second stage will produce (25%) of the data in the first stage. Thus, GTTP will send about (6.25%) of original data, which results in a significant increase in the network performance. It is hypothesized that the proposed study will minimize impairment in transmitting genomic big data using GTTP, as well as improve network throughput by sending less data. Therefore, GTTP represents a new strategy in transfer protocol literatures, considers scope, kind, and size of data. Also, transmitting 6.25% of the original data increases network throughput and decreases traffic latency, rather than reduces transfer time.

5.3.1 Experimental Results

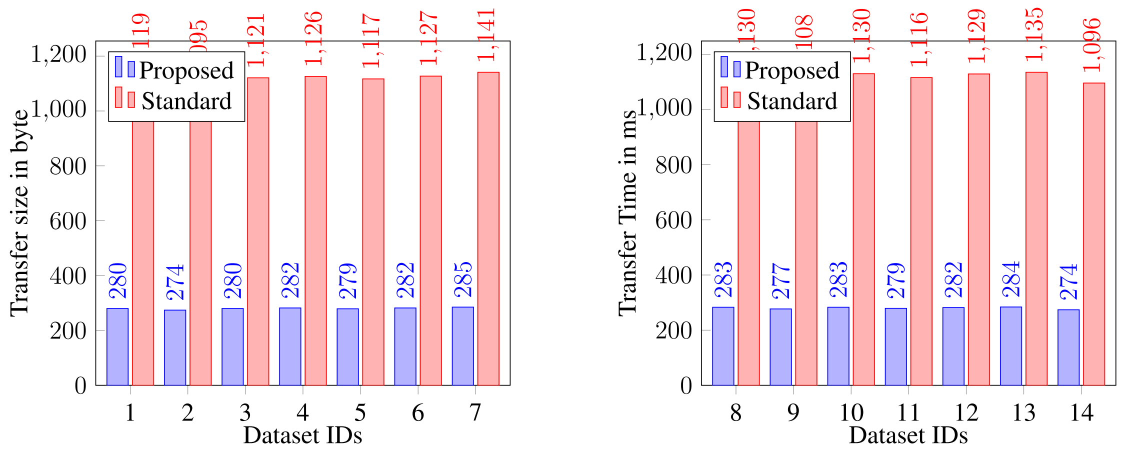
In order to validate our theoretical results, we performed (28) experiments on different genomic data for up to 10GB. We used (FASTA format) as genomic files to exchange between two machines (client-server). Experimental results can be seen in Figures 5.6 on page 107, 5.7 on page 108, and 5.8 on page 108. Those results came from our implementation using visual C# language version (2013) on the following machine specifications: Windows 8.1 pro (64-bit), Intel Core i7 with clock speed of 2.4 GHz. The system is equipped with 8GB RAM, L2 Cache (per Core) of 256 KB and L3 Cache of 6 MB over 37.59 Mb/s DOWNLOAD and 4.22 Mb/s UPLOAD speeds.



(a) Transfer size of FASTA datasets: 1 - 7 (b) Transfer size of FASTA datasets: 8 - 14

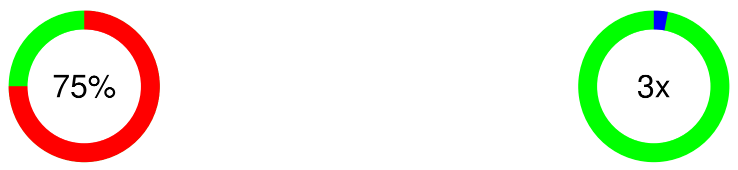
Figure 5.6: Transfer size of real FASTA datasets 1 - 14 using proposed and standard data-encoding approaches

All tested files came from The National Center for Biotechnology Information Ad-



(a) Transfer time of FASTA datasets: 1 - 7 (b) Transfer time of FASTA datasets: 8 - 14

Figure 5.7: Transfer time of real FASTA datasets 1 - 14 using proposed and standard data-encoding approaches



(a) Size reduction of FASTA (b) Time acceleration of FASTA

● Proposed (time/size) ● Standard (size) ● Standard (time)

Figure 5.8: The maximum size reduction and time acceleration of tested genomics datasets

vances Science and Health (NCBI) [117] except 1GB and 10GB, which we generated randomly by using a special genomic generator for FASTA files.

Our experiments start by requesting a specific genomic data file by client from the server. The server encodes the required data into binary form and passes them to the GZIP algorithm to send a zipped file to the client. When a client receives a zipped file, the client starts decompress and decodes it to get original data.

Also, we implemented a normal HTTP request-response to compare and assess with our GTTP results. We sent and received genomic files with the traditional HTTP model and then compared the size and transfer time with our new approach. Experiments were performed on different genomic files to determine the minimum time required to transfer them using both protocols,

Table 5.7: Size reduction and time acceleration of datasets

Dataset	Size Reduction	Time Acceleration
	Proposed vs. Standard	Proposed vs. Standard
1	76%	75x
2	75%	75x
3	76%	75x
4	76%	75x
5	76%	75x
6	76%	75x
7	76%	75x
8	75%	75x
9	75%	75x
10	76%	75x
11	75%	75x
12	77%	75x
13	75%	75x
14	76%	75x
15	75%	75x
16	76%	75x
17	76%	75x
18	75%	75x
19	76%	75x
20	75%	75x
21	75%	75x
22	77%	75x
23	75%	75x
24	76%	75x
25	76%	75x
26	76%	75x
27	76%	75x
28	75%	55x

HTTP and GTTP. Determining the minimum time to transfer genomics is a way of assessing the encoded scheme. The GTTP encoded mechanism decreased the size of the transmitted data, which also reduced the transfer time. Figure 5.6 on page 107 shows transferring genomics size via both protocols, HTTP and GTTP. For example, experiment 2 transfers 10,203,125 bytes using HTTP, while we only need to transfer 2,508,540 bytes of data using GTTP, which is a 75% decrease in the data set size. Also, experiment 27 transfers 1,103,390,067 bytes (1 GB) through HTTP, whereas 264,788,421 bytes are transferred (0.25 GB) through GTTP, thereby saving 0.76 to send 0.24 instead. Figure 5.7 on page 108 shows transferring time for genomic files using both protocols HTTP and GTTP. For instance, HTTP needed 1,095 ms to transfer 10,203,125 bytes from Experiment 2, whereas GTTP spent only 274 ms to transfer the same file 3.99 times faster. Also, HTTP spent 2,474,232 ms to transfer 10,737,418,240 bytes (10 GB) from experiment 28, while GTTP spent only 1,104,290 ms to transfer the same file, which is 2.24 times faster.

Briefly, our implementation results indicate a big saving of data and increasing in transfer speed as presented in Figure 5.8 on page 108. Thus, using GTTP reduces the amount of transmitted data and increases the network throughput.

5.4 Conclusions

This chapter presents the design and implementation of a data-aware variable-length and naive bit models of content-encoding techniques to reduce big genomic datasets during data transfer process. Our protocol exploits the fact that genomic data is limited in its alphabet and is largely redundant. This fact allows us to design a variable length encoding scheme which decreases significantly the size of the genomic data that needs to be transferred over the network. Consequently, an enormous reduction in the time is also observed as compared to traditional HTTP and FTP protocols. Our results also show that by using the proposed encoding schemes, the resulting protocol that uses a single machine is better than 10 machines that use the traditional HTTP protocol to transfer genomic data.

CHAPTER 6

A NEW CRYPTOGRAPHY ALGORITHM TO PROTECT CLOUD-BASED HEALTHCARE SERVICES

6.1 Introduction

The revolution of smart devices has a significant and positive impact on the lives of many people, especially in regard to elements of healthcare. In part, this revolution is attributed to technological advances that enable individuals to wear and use medical devices to monitor their health activities, but remotely. Also, these smart, wearable medical devices assist health care providers in monitoring their patients remotely, thereby enabling physicians to respond quickly in the event of emergencies. An ancillary advantage is that health care costs will be reduced, another benefit that, when paired with prompt medical treatment, indicates significant advances in the contemporary management of health care. However, the competition among manufacturers of these medical devices creates a complexity of small and smart wearable devices such as ECG and EMG. This complexity results in other issues such as patient security, privacy, confidentiality, and identity theft. In this chapter, we discuss the design and implementation of a hybrid real-time cryptography algorithm to secure lightweight wearable medical devices.

The proposed system is based on an emerging innovative technology between the genomic encryptions and the deterministic chaos method to provide a quick and secure cryptography algorithm for real-time health monitoring that permits for threats to patient confidentiality to be addressed. The proposed algorithm also considers the limitations of memory and size of the wearable health devices. The experimental results and the encryption analysis indicate that the proposed algorithm provides a high level of security for the remote health monitoring system.

The purpose of this chapter is to design and implement a hybrid cryptography algorithm to protect cloud-based health services. Moreover, it will introduce a generic concept that can be used by other cloud-based applications to secure data that exchange remotely.

6.1.1 Contribution

This chapter discusses the design and implementation of a novel cryptography algorithm for the remote, real-time health monitoring services and relies on a combination of the genomics-based encryption, and the deterministic chaos method. This work aims to secure the health data of cloud-based services between wearable medical devices attached to the patients, and accessible to healthcare providers such as physicians, as well as hospitals. We assert that creating a new cryptography mechanism for the wearable health devices can solve the delay issue of complex computations of traditional encryption approaches.

6.1.2 Motivation

It is not a minor challenge to implement a new cryptography algorithm by combining two different techniques. Also, the high demand on small, friendly, and affordable wearable health devices encourages manufacturing production. However, producing wearable devices to provide many services remotely, such as cloud-based health services, creates security challenges. Data security and privacy are very important to both users and service providers, especially for remote healthcare services. Many solutions have been developed for challenges of remote diagnostic devices. However, security and cryptography challenges have not been addressed at the same level, mainly due to compatibility issues. As a result, scientists are motivated to navigate and search for new methods to transfer health data more efficiently and in a fully secured environment.

Current data encryption methods are not suitable for the cloud-based services such as remote healthcare monitoring due to using heterogeneous devices that use a variety of transfer protocols that belong to different vendors. Observing these facts, we take advantage of the nature of the genomic encryption and the deterministic Chaos Theory to implement a more efficient cryptography algorithm to secure remote healthcare monitoring.

6.1.3 Chapter Goals and Organization

The purpose of this chapter is to design and implement a hybrid cryptography algorithm to protect cloud-based health services. Moreover, it will introduce a generic concept that can be used by other cloud-based applications to secure data that exchange remotely. The contributions of this chapter are outlined as follows:

- Summarizes the genomic encryption method and Chaos Theory with quick fundamentals, sans the need to search through the details presented in the standards' specifications.
- Provides an overview of the challenges of the wireless sensing devices in terms of security and computations.
- Presents the need for better cryptography methods to provide better cloud-based health services.

The remainder of this chapter is organized as follows: Section 6.2 presents preliminaries of this work for both: genomic encryption method and the Chaos Theory. Section 6.3 discusses the overall architecture of the proposed algorithm and model description. Section 6.4 presents the experiments and results of the proposed cryptography algorithm. Section 6.5 discusses the comparative analysis and Section 6.6 presents our conclusions.

6.2 Preliminaries

6.2.1 Genomic-Based Cryptography

Deoxyribo Nucleic Acid (DNA) is a biochemical macromolecule that contains genetic information necessary for the functioning of living beings. The genomics include the entire hereditary information about the organism cell and consists of the chromosomes in a cell's nucleus and components of the genome. A genomic molecule consists of a double-stranded nucleotides structure that is obtained by two twisted single-stranded DNA chains, hydrogen bonded together between bases (A-T and G-C) [118]. The double-helix structure is configured by two single, antiparallel strands.

Four kinds of bases are found in two strands: **Adenine (A)**; **Guanine (G)**; **Thymine (T)**; and **Cytosine (C)**. A strand contains a sequence of bases in a specific pattern. The other strand contains the complementary nucleotides of the first strand. The adenine pairs with a thymine by using a double bond (A = T), while the thymine and the cytosine pair with each other by using a triple bond (G = C). The genomic sequencing information is contained in the nucleotide bases.

Genomic-based cryptography emerged as a new cryptographic field in 1994 and has been used as an information carrier and as a modern implementation tool in biological technology. The computational process using genomic-based cryptography produces a sequence of nucleotides: A, T, C, and G, as the encrypted data output. The genomic-based cryptography is done by hybridization of the DNA molecules and is formed by a double helix structure of complementary base pairs to encode data. The DNA addition and subtraction operation rules, described in Tables 6.1 and 6.2 on page 114, are used to confuse the ECG data values, such that A= 00; T= 01; C= 10; G= 11.

Table 6.1: Subtraction operation for the DNA sequence

-	A	T	C	G
A	A	G	C	T
T	T	A	G	C
C	C	T	A	G
G	G	C	T	A

6.2.2 Deterministic Chaos Theory

Chaos functions have mainly been used to develop mathematical models of nonlinear systems. They have attracted the attention of many mathematicians owing to their extremely sensitive nature of initial conditions, as well as their enormous applicability to modeling complex problems of daily life. The sequences produced by such functions have very good randomness and

Table 6.2: Addition operation for the DNA sequence

+	A	T	C	G
A	A	T	C	G
T	T	C	G	A
C	C	G	A	T
G	G	A	T	C

complexity. These functions have an extreme sensitivity to initial conditions. For example, if the initial start value of a chaotic function is modified 10^{-20}, iterative numbers produced after some iterations appear to differ from each other. This extreme sensitivity to initial conditions and some other interesting properties, such as pseudo-randomness, wide spectrum, and good correlation indicate that chaotic functions may serve as a promising alternative to conventional cryptographic algorithms.

The main advantage using Chaos Theory lies in the observation that a chaotic signal looks like noise to unauthorized users. Moreover, generating chaotic values is often low cost with simple iterations, which makes it suitable for the construction of stream ciphers. Therefore, cryptosystems can provide a secure and fast means of data encryption, which is crucial for data transmission in many applications. Generally speaking, chaotic stream ciphers use chaotic systems to generate pseudorandom key streams to encrypt the data (one-by-one). In this work, a 1-D tent logistic map is employed for key generation. The map generates chaotic sequences in the interval [0, 1], assuming the following equation [119]:

$$x_{n+1} = \begin{cases} \frac{x_n}{\mu} & 0 \leq x_n \leq \mu \\ \frac{1-x_n}{1-\mu} & \mu < x_n \leq 1 \end{cases} \tag{6.1}$$

where x_n refers to the state variable of the system that belongs to the interval [0, 1], $\mu \in$ (0,1], and $x_0 \in$ [0,1] is the initial condition. A typical orbit with initial condition $x_0 = 0.6$ and control parameter $\mu = 0.8$ is shown in Figure 6.1 (a) on page 117. The distribution of the points of the orbit with a length of 1000 points is shown in Figure 6.1 (b) on page 117. The histogram plot is a graphical representation similar to the bar chart where it shows the distribution of data with ranges of the data grouped into intervals.

6.2.3 One-Time Pad Encryption Method

The one-time pad encryption mechanism is simple: encrypt each sample of the data by the addition modular, which gets a bit or character from a random key generator. For example, the key sequence generated by a random generator are as following equations:

$$pad = k_i = k_1, k_2, k_3, ..., k_n, \quad k_i \in [0, 1]. \tag{6.2}$$

The original message which will be encrypted by the pad keys is as follows:

$$message = m_i = m_1, m_2, m_3, ..., m_n, \quad m_i \in [0, 1]. \tag{6.3}$$

Then the cipher is as follows:

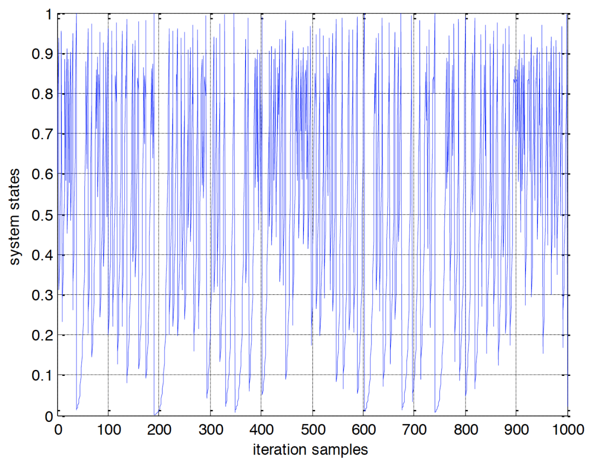
$$c_i = m_i \oplus k_i. \tag{6.4}$$

To decrypt the cipher in the receiver side, the following function is used:

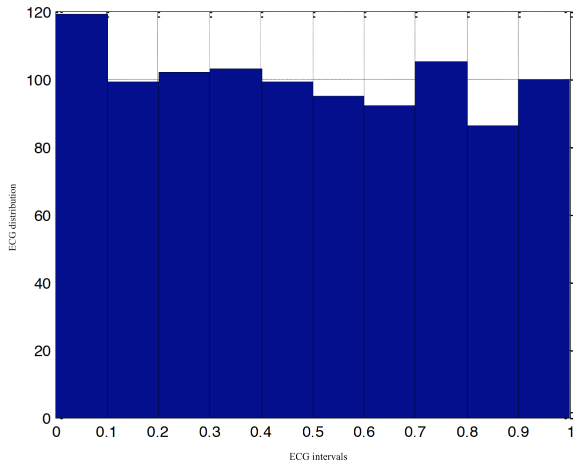
$$m_i = (m_i \oplus k_i) \oplus k_i. \tag{6.5}$$

6.3 Proposed Encryption Algorithm

This section provides step-by-step details of the proposed algorithm for the data encryption by combining a DNA-based encryption technique and chaotic logistic maps. Limitation of the wearable sensor node memories causes each sample of the ECG signal to be encrypted alone. The sample data is a 16-bit length (the most 4-bit are considered to be zeros when the 12-bit



(a) The chaotic orbit for parameters of $x_0 = 0.6, \mu = 0.8$



(b) Histogram of the points of typical orbit with 1000 points length

Figure 6.1: The chaotic orbit and the histogram of the orbit with 1000 points length

analog to digital converter is used). Each sample is divided and encoded into 8 DNA bases, with each consisting of a two-bit length. A chaotic sequence is generated by using a tent chaotic map with the initial condition xo1 and constant parameter μ_{01}. The sequence is scaled to [0, 65536] and converted into 8 DNA bases, with the most 4-bit equating to zeros. The 8 new DNA bases are generated by combining the encoded ECG sample and encoded chaotic random key, as shown in Tables 6.1 and 6.2 on page 114.

Two of 1-D tent chaotic maps with different initial conditions x_{02} and x_{03} and constant parameters μ_{02} and μ_{03} are used to construct the 2-D encoding matrix, which will be used as a complement (or not) to the combination of the ECG signal and the first chaotic map. After that, we will obtain the six encryption keys: x_{01}, x_{02}, x_{03}, μ_{01}, μ_{02}, and μ_{03}. The computational precision of the 64-bit double precision number is 2^{52}, according to the IEEE standard for the floating-point arithmetic(IEEE 754). Therefore, the total number of different keys used is $(2^{52})^6 = 2^{312}$. Such a large key space is efficient and sufficient for reliable practical use.

The proposed algorithm of the ECG sensory data was initially introduced by the work of [62] and [68], where they used it for image encryption. A modification of this algorithm has been made to be applicable for WWBASN, such that each sample of the ECG data is encrypted alone, since it works on limited resources in terms of memory and computation. The proposed algorithm works in 10 simple steps as follows:

Step 1:

Convert the signal samples into a binary sequence as a $n * m$ binary matrix (m = 16 bits).

Step 2:

Encode the binary sequence into a matrix of nucleotides (DNA sequence matrix) to get the encoding matrix $\frac{n*m}{2}$ such that: A = 00, T = 01, C = 10, G = 11.

Let's assume the signal data vector

$$s = \begin{pmatrix} 2055 \\ 1250 \\ 3590 \end{pmatrix}$$

Then the binary sequence would be:

$$s = \begin{pmatrix} 0000100000000111 \\ 0000010011100010 \\ 0000111000000110 \end{pmatrix}$$

So, we get the DNA sequence as follows:

$$s = \begin{pmatrix} AACAAATG \\ AATAGCAC \\ AAGCAATC \end{pmatrix}$$

Step 3:

Divide the DNA sequence matrix into 8 sub-matrices each $\frac{n*m}{8}$ as follows:

DNA sub-matrix$_1$ is $s_1 = \begin{pmatrix} A \\ A \\ A \end{pmatrix}$, DNA sub-matrix$_2$ is $s_2 = \begin{pmatrix} A \\ A \\ A \end{pmatrix}$

DNA sub-matrix$_3$ is $s_3 = \begin{pmatrix} C \\ T \\ G \end{pmatrix}$, DNA sub-matrix$_4$ is $s_4 = \begin{pmatrix} A \\ A \\ C \end{pmatrix}$

DNA sub-matrix$_5$ is $s_5 = \begin{pmatrix} A \\ G \\ A \end{pmatrix}$, DNA sub-matrix$_6$ is $s_6 = \begin{pmatrix} A \\ C \\ A \end{pmatrix}$

DNA sub-matrix$_7$ is $s_7 = \begin{pmatrix} T \\ A \\ T \end{pmatrix}$, DNA sub-matrix$_8$ is $s_8 = \begin{pmatrix} G \\ C \\ C \end{pmatrix}$

Step 4:

Generate a chaotic sequence vector n1 through a 1-D chaotic map with initial condition x_{01} and constant parameter μ_{01}. Scale the chaotic sequence to [0, 65536].

Step 5:

Apply steps 1 to 3 to the chaotic sequence.

Step 6:

Add the DNA sub matrices of the original signal to the DNA sub matrices of the chaotic sequence, according to the rules shown in Tables 6.1 and 6.2 on page 114.

Step 7:

Recombine the sub matrices generated from the **Step 6** to form a new binary sequence matrix $c(n * m)$.

Step 8:

Generate two chaotic sequences, $c_1(n * 1)$ and $c_2(1 * m)$, along with another initial condition x_{02}, x_{03}, and constant parameters μ_{02} and μ_{03}. After that, multiply the two vectors to produce a matrix $w(n * m)$ with range [0, 1]. Map the value of w into (0, 1) by mod (w, 1). Then use the following threshold function $f(x)$ to get the binary sequence matrix:

$$f(x) = \begin{cases} 0, & 0 < w(i,j) \leq 0.5 \\ 1, & 0.5 < w(i,j) \leq 1 \end{cases}, \tag{6.6}$$

Step 9:

Get the complement to the matrix $w(i, j) = 1$, then $c(i, j)$ is complemented. Otherwise, it is

unchanged. For example: if the first row of the
w is $\begin{pmatrix} 1001100110111001 \end{pmatrix}$,
c is $\begin{pmatrix} 1010100100101010 \end{pmatrix}$,
c' is $\begin{pmatrix} 0011000010010011 \end{pmatrix}$
This would produce a new encoding matrix that is $c'(n * m)$. Rescaling this matrix produces the encrypted ECG sample, which would be transmitted through the wireless channels.

Step 10:

Apply the inverse process of **Step 2** and **Step 1** for the sequence matrix c', then we will get a real value of the matrix **D** that represents the encrypted data signal.

In the decryption process, the reverse steps are applied from the **Step 10** to the **Step 1**. Also, use the subtraction operation instead of the addition operation as shown in Tables 6.1 and 6.2 on page 114.

6.4 Experiments and Results

In this work, we used the SHIMMER platform [120] as the embedded sensor system. From the hardware viewpoint, this platform includes the following:

A low-power 16-bit microcontroller (Texas Instrument MSP430F1611), a low-power radio supported with 802.15.4, an extension module for the ECG, **E**lectro**M**yo**G**raphy (EMG), and **G**alvanic **S**kin **R**esponse (GSR), and built-in triaxial accelerometer.

The MSP430 microcontroller runs at 8 MHz, has 10 KB of RAM, 48 KB of flash memory, and includes a fast hardware multiplier. As a practical implementation, five sensor nodes were used as a body area network, which includes the ECG, EMG, and accelerometers in the human chest, thigh, and leg. The chest node was used as a coordinator for the body area network, while the encryption algorithm was implemented only inside the ECG node. The nodes were programmed using the TinyOs, and the data were transmitted to the base station node using Python 2.7 under a Linux operating system to decrypt the data. There is a significant problem that effects

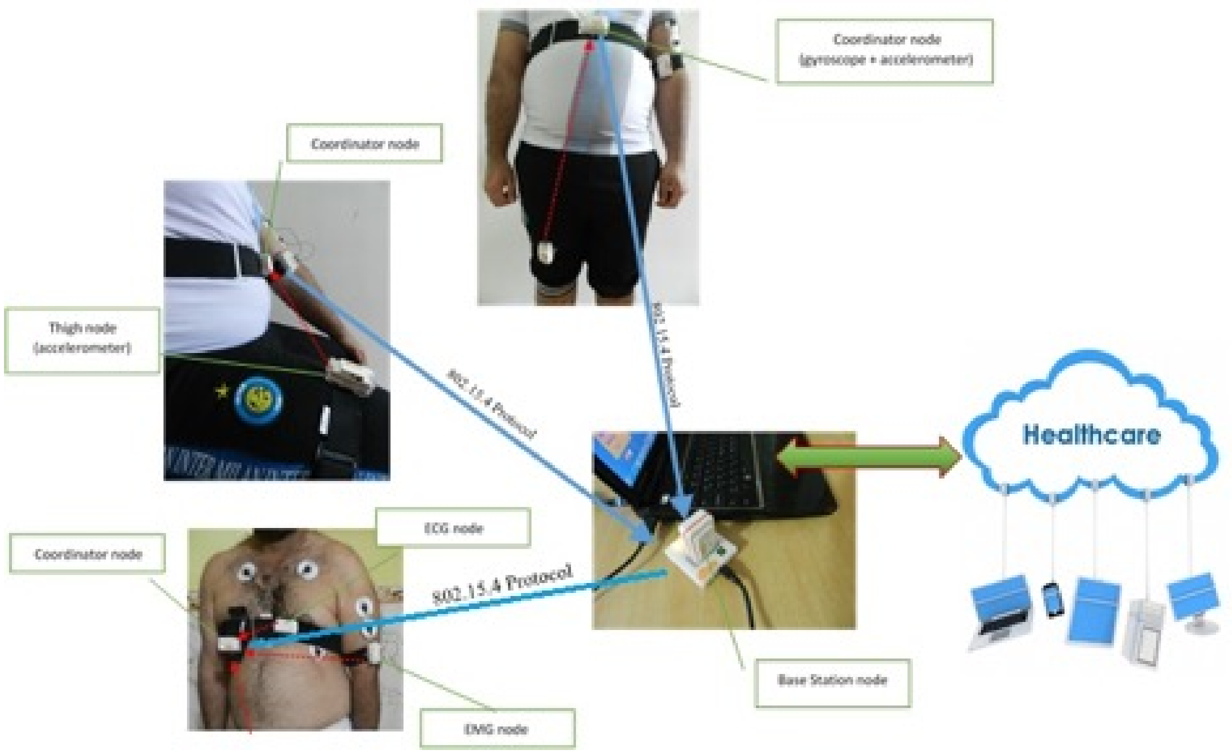
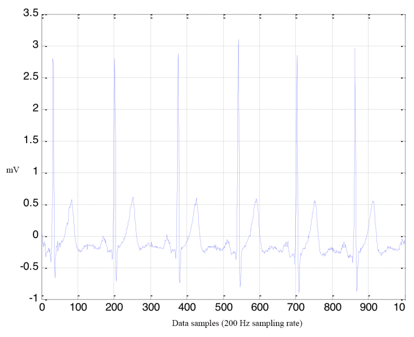


Figure 6.2: Architecture of the proposed system

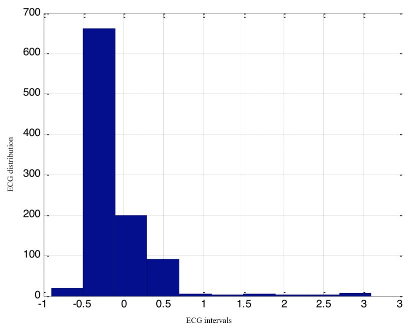
the performance of the decryption process by reducing the transmitted packets in the time unit when using a collision-free (MAC) protocol.

The proposed algorithm was implemented and tested using a SHIMMER sensor nodes platform, a python programming language, and a TinyOs 2.1.2 that supports the floating-point computation. The proposed system is designed to support multiple patients (up to 256), using a single base-station node. Also, it can support different groups of patients: each group belongs to a different base-station node. In this mode of the operation, different scenarios could be considered, such as intensive care unit, hospital rooms, rehabilitation units, or patient homes. The proposed system has been designed to be secure, scalable, effective, and easy to use, as shown in Figure 6.2 on page 121. The sensory data packet from different types of sensor nodes, which are attached to the patient's skin, will send information to the coordinator node, using the IEEE 802.15.4 protocol. The coordinator node will forward the packets to the base-station node, using the same protocol.

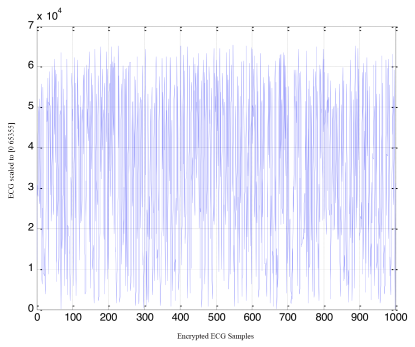
The original ECG sensed signal is shown in Figure 6.3 (a) on page 122, while the



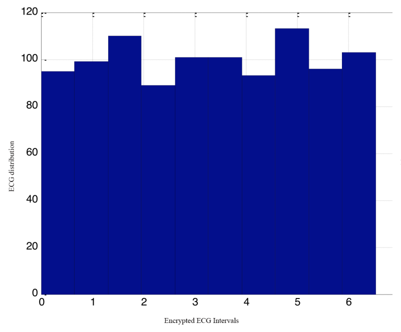
(a) Original ECG signal



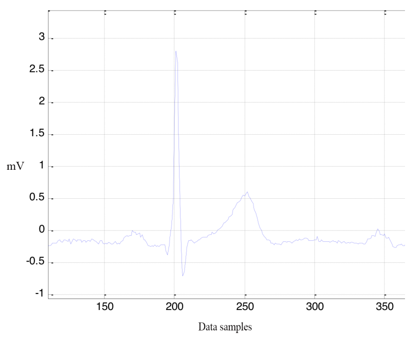
(b) Histogram of the original ECG signal



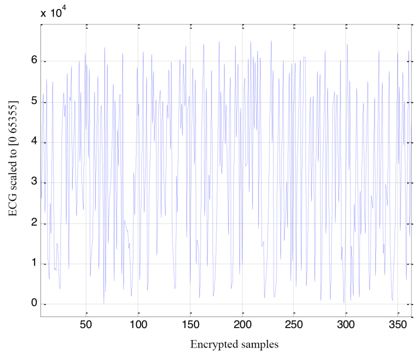
(c) Encrypted ECG signal



(d) Histogram of the encrypted ECG signal

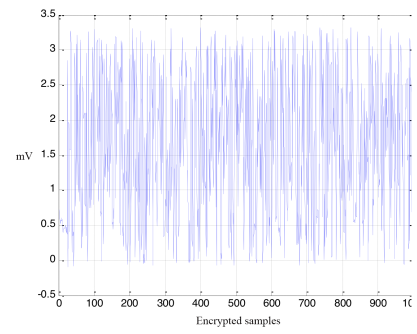


(e) Single beat ECG signal

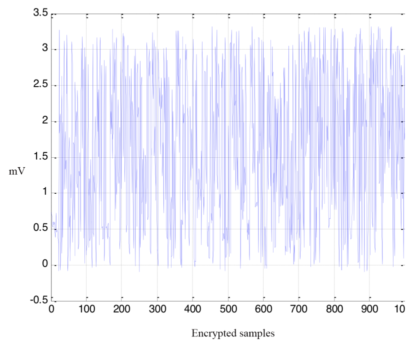


(f) Encrypted of the single beat

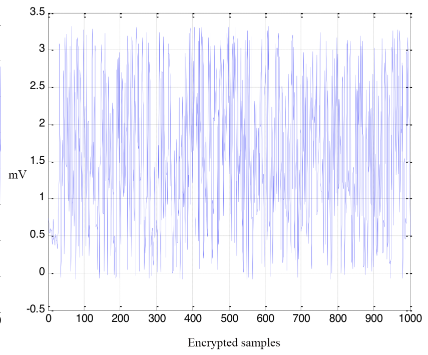
Figure 6.3: Different decrypted ECG signal with different key values of the first group



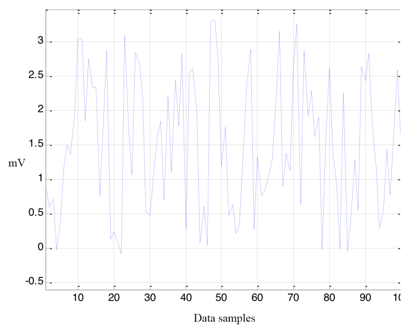
(a) $x_{01} = 0.20000000000000001, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.66, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$



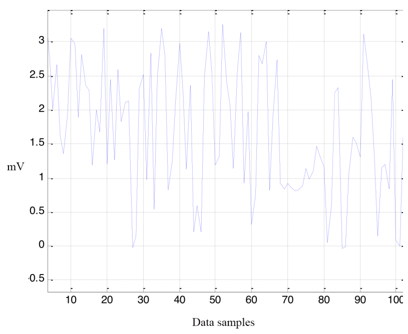
(b) $x_{01} = 0.2, x_{02} = 0.41000000000000001, x_{03} = 0.61, \mu_{01} = 0.66, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$



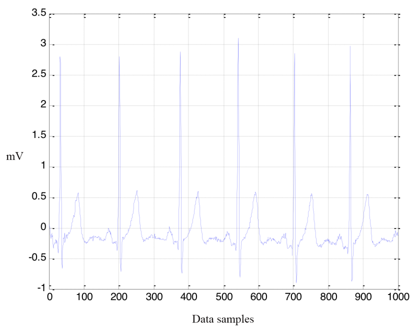
(c) $x_{01} = 0.2, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.66000000000000001, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$



(d) $x_{01} = 0.9, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.66, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$



(e) $x_{01} = 0.5, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.3, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$



(f) Decrypted ECG signal with exactly the same key values $x_{01} = 0.2, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.3, \mu_{02} = 0.3$, and $\mu_{03} = 0.99$

Figure 6.4: Different decrypted ECG signal with different key values of the second group

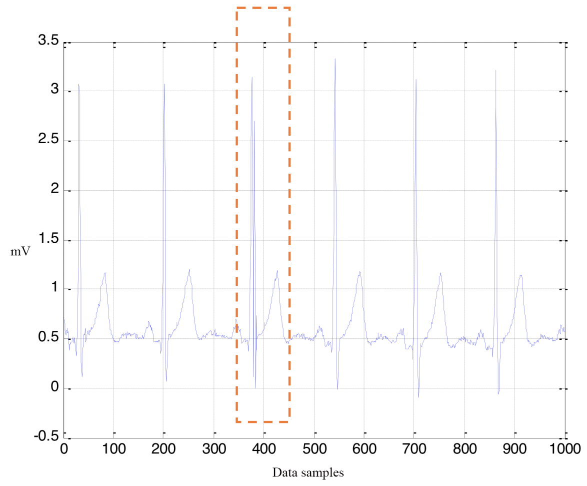
histogram of the original signal is shown in Figure 6.3 (b) on page 122, where most of the ECG data are between -0.5 and 0.5 mV. The Figure 6.3 (c) on page 122 is shown the encrypted ECG signal according to the proposed encryption algorithm, where the signal is detected as a noise due to the nature of the chaos function. The key values used were $x_{01} = 0.2, x_{02} = 0.41, x_{03} = 0.61, \mu_{01} = 0.66, \mu_{02} = 0.4$, and $\mu_{03} = 0.99$. Figure 6.3 (d) on page 122 is depicts the histogram of the encrypted ECG signal. Here, the contribution of the encrypted data is uniform in contrast to the Figure 6.3 (b) on page 122. The single beat ECG signal is illustrates in Figure 6.3 (e) on page 122, while Figure 6.3 (f) on page 122 displays a zoom window of the decrypted signal. Figure 6.4 (a, b, and c) on page 123 show different changes in key values that demonstrate the power of the proposed algorithm. This is because of the sensitivity of the chaos function to any changes in the initial conditions. The decrypted signals that use the same encryption keys are shown in Figure 6.4 (d) on page 123, while Figure 6.4 (e and f) on page 123 shows a sample window of the encrypted ECG signal after changing the initial condition x_{01}. These figures show the difference of the initial values of the decrypted signals. These values are as follows : $\frac{x_{01}}{\mu_{01}}$ and use the Equation 6.6 on page 119.

Figure 6.5 (a) on page 125 reveilles the real-time encryption process when sending the ECG sensor node packets to the body area network. Figure 6.5 (b) on page 125 depicts a zoom around the 380^{th} samples, where the packets were lost. The decryption process assumes these samples ($\sim$20 successive samples) are zeros. The whole decryption process was not affected by the lost samples since the encryption process decrypts each sample alone. The fluctuation shown in the loss. occurs due to the chaotic nature of the decryption process.

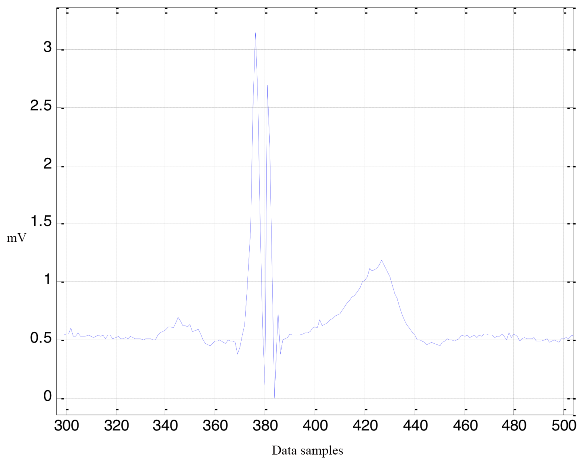
The correlation coefficients are important features, and they are calculated based on the correlation between the encrypted and the original signals. The main points that can be obtained from the correlation coefficients are listed as follows:

1. When it is close to the value of **1**, then there is a positive linear relationship between the two vectors.

2. When it is close to the value of **-1**, then there is a negative linear relationship between the



(a) ECG signal with some lost samples



(b) Zoom window near the 380^{th} sample shows the effect of samples loss

Figure 6.5: Zoomed of encrypted ECG signals

two vectors.

3. When it is close to the value of **0**, then there is no linear relationship between the two vectors.

The following equations are used to determine the correlation coefficient [121]:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{6.7}$$

$$D(x) = \frac{1}{N}\sum_{i=1}^{N} (x_i - E(x))^2, \tag{6.8}$$

$$cov(x) = \frac{1}{N}\sum_{i=1}^{N} (x_i - E(x))(y_i - E(y)), \tag{6.9}$$

$$r_{xy} = \frac{cov(x,y)}{\sqrt{D(x)}\sqrt{D(y)}}. \tag{6.10}$$

The correlation coefficient between the decrypted ECG signal and the original ECG signal was 1, and 0.0215 between the original ECG and the encrypted signals.

6.5 Comparative Analysis

Since wearable wireless sensor networks have limited storage and computational resources, most of the encryption algorithms are not feasible on these platforms. Connecting this network to the Internet makes the system more vulnerable, since it would be easy for an intruder to gain access to the patients' data, especially in case the physical distance would not be a problem. Hence, a powerful encryption algorithm is required to maintain the privacy of patient data with the limited resources of such a network.

There are many effective crypto-systems of traditional encryption algorithms used for information security, such as **D**ata **E**ncryption **S**tandard (DES); **T**riple DES (3DES); Blow fish; and **A**dvanced **E**ncryption **S**tandard (AES). DES suffers from the key size that is (**56**-bit), and it uses a **64**-block ciphering. There are some potential issues that can occur, especially when encrypting several gigabytes of data using the same key. The use of the 3DES enables the reuse of the DES implementation by cascading three instances of DES (with distinct keys). This algorithm is secure up to **2186** key spaces, but it is slow. Blow fish is a symmetric block cipher that uses a variable length of a key between **32** to **448** bits. It uses key-dependent lookup tables. Hence, performance depends on the resources that are available with the platform used. AES accepts keys of **128**; **192** and **256**-bit length, and uses **128**-bit blocks. It is an efficient algorithm from the software and hardware perspectives [122].

Table 6.3 on page 127 summarizes different encryption algorithms in terms of key length, block ciphering, number of keys, and applicability in a wireless sensor network. A real-time DNA-based encryption algorithm and Chaos Theory for secure healthcare information, has been proposed for the ECG signal encryption. The proposed algorithm has been designed to be a context-aware algorithm, where all computations are performed on the sensor node. The algorithm was designed to work with limited resources of the computational sensor nodes, where other encryption methods cannot be implemented with these resources.

The use of Chaos Theory, as a key generator, is more powerful than the pseudorandom generator. The DNA-based encryption is a solid approach that can be used for data encoding, since it requires a minimum of computations. Furthermore, the nature of the one-time pad to encrypt each sample individually makes it an appropriate technique to be used in a wireless sensor network, where it minimizes the required memory space. Finally, the algorithm has been implemented successfully in a real-time body area network for healthcare monitoring and is suitable to be used in the presence of a collision in wireless communication. For future work, it is important to develop an algorithm to predict the sample loss and to design a collision-free MAC (media access control) protocol.

Table 6.3: Encryption algorithm comparisons

Algorithm	Key space (bit)	Number of keys (bit)	Block cipher	Applicable in WSN
DES	56	1	yes	no
3DES	168	3	yes	no
Blow fish	32-448	huge	yes	no
AES	128, 192, 256	1	yes	no
Proposed	312	each sample has it's a unique key	no	yes

6.6 Conclusions

Smart connected wearable medical devices have gained the attention of both patients and professional healthcare providers due to several characteristics, such as the fact that they are lightweight, small in size, easy to use, and inexpensive. Also, the development of communication protocols and techniques, such as cloud-based services, enables many healthcare providers to deliver their services to patients remotely. However, the remote-based and cloud-based services have security issues, due to connecting heterogeneous devices that come from different vendors. Use of traditional cryptography algorithms does not protect health data over the network because the data are transferred through multiple devices and protocols.

In this chapter, we designed and implemented a novel encryption algorithm that relies on the utilization of genomics encryption and deterministic chaos to protect the remote healthcare monitoring system. The proposed algorithm protects the health data from the main threats, such as a key theft, man-in-the-middle attack, and brute force attack. Also, the proposed algorithm is designed in a way that considers the limitations of device size, memory capacity, power consumption, and cost. Moreover, the proposed algorithm does not require complex computations to encrypt the data. The practical implementation of the proposed encryption algorithm and result analysis prove that it is ideally suitable for remote health monitoring services in terms of data security, computations, and power consumption.

CHAPTER 7
CONCLUSIONS

In this dissertation, we implemented a novel deep learning-based data minimization algorithm to integrate with transfer protocols to reduce the size of big genomic datasets during the transfer phase, and then to transfer the data securely in less time. Also, we implemented three other data encoding techniques to reduce big genomic datasets during data transfer processes. The implementation results illustrate that the proposed data minimization algorithm is capable of reducing the transfer time 99-fold, compared to the standard content-encoding of HTTP, and 96-fold compared to FTP on tested datasets. We used GZIP and MFCompress algorithms as optional compression algorithms, in addition to our data minimization algorithm to assess how the transfer protocol behaves in terms of transfer time and size. Also, we showed that our data minimization algorithm provides the best size reduction, reduces transfer time, and securely transfers big genomic datasets. Our proposed data minimization mechanism relies on a deep learning-based method, while encoding the data during data transfer, and then transfers the data securely in a shortened time, as illustrated in section 3.3.3 on page 48. We demonstrated that the data size can be significantly reduced by our adaptive encoding, compared to a standard content-encoding scheme, with and without compression algorithms, as well. Also, we implemented a genomic dataset generator of a FASTA file format to verify the performance of our current data minimization scheme and then compared it to the standard, as well as our previous content-encoding schemes. Our proposed genome generator allowed us to control the repetition in the data, which was instrumental in assessing the performance of our data minimization algorithm. The tested encoding schemes, standard and proposed, were implemented over HTTP, FTP and BitTorrent protocols, with/out involving compression algorithms. Our experiments indicated that utilizing a CNN-based content-encoding scheme performs much better than the current and common use of the transfer protocol content-encoding scheme by assigning short codewords for the dataset characters. We conclude that the proposed data minimization algorithm provides the best performance among current content-encoding approaches for big genomic datasets.

In chapter 7, we designed and implemented a novel encryption algorithm that relies on the utilization of genomics encryption and deterministic chaos to protect the remote healthcare monitoring system. The proposed algorithm protects the health data from the main threats, such as a key theft, man-in-the-middle attack, and brute force attack. Also, the proposed algorithm is designed in a way that considers the limitations of device size, memory capacity, power consumption, and cost. Moreover, the proposed algorithm does not require complex computations to encrypt the data. The practical implementation of the proposed encryption algorithm and result analysis prove that it is ideally suitable for remote health monitoring services in terms of data security, computations, and power consumption.

REFERENCES

[1] C. Mora, D. P. Tittensor, S. Adl, A. G. Simpson, and B. Worm, “How many species are there on earth and in the ocean?” *PLoS biology*, vol. 9, no. 8, e1001127, 2011.

[2] J.-Y. Li, J. Wang, and R. S. Zeigler, “The 3,000 rice genomes project: New opportunities and challenges for future rice research,” *GigaScience*, vol. 3, no. 1, p. 8, 2014.

[3] F. S. Collins and H. Varmus, “A new initiative on precision medicine,” *New England Journal of Medicine*, vol. 372, no. 9, pp. 793–795, 2015.

[4] T. C. Carter and M. M. He, “Challenges of identifying clinically actionable genetic variants for precision medicine,” *Journal of healthcare engineering*, vol. 2016, 2016.

[5] N. Drake *et al.*, “Cloud computing beckons scientists.,” *Nature*, vol. 509, no. 7502, pp. 543–544, 2014.

[6] R. Spencer, “The square kilometre array: The ultimate challenge for processing big data,” in *Data Analytics 2013: Deriving Intelligence and Value from Big Data, IET Seminar on*, IET, 2013, pp. 1–26.

[7] M. C. Schatz, “The next 20 years of genome research,” *Simons Center for Quantitative Biology*, 2015.

[8] M. Aledhari, M. Di Pierro, T. Barkman, M. Hefeida, and F. Saeed, “Deep learning-based data minimization algorithm to fast and securely transfer of big genomic datasets,” *IEEE TRANSACTIONS ON BIG DATA*, 2017.

[9] J. Hadfield and N. Loman, *Next generation genomics: World map of high-throughput sequencers*, 2014.

[10] A. Regalado, “Emtech: Illumina says 228,000 human genomes will be sequenced this year,” *Technology Review*, vol. 24, 2014.

[11] S. C. Baker, “Next-generation sequencing challenges,” 2017.

[12] A. Manzoor, “Emerging role of big data in public sector,” *Managing Big Data Integration in the Public Sector*, pp. 268–88, 2015.

[13] J. Zhu, “A year of great leaps in genome research,” *Genome medicine*, vol. 4, no. 1, p. 4, 2012.

[14] N. I. of Health *et al.*, “An overview of the human genome project,” 2005.

[15] R. A. Gibbs, J. W. Belmont, P. Hardenbol, T. D. Willis, F. Yu, H. Yang, L.-Y. Ch’ang, W. Huang, B. Liu, Y. Shen, *et al.*, “The international hapmap project,” 2003.

[16] W. S. Bush and J. H. Moore, “Genome-wide association studies,” *PLoS computational biology*, vol. 8, no. 12, e1002822, 2012.

[17] M. D. Mailman, M. Feolo, Y. Jin, M. Kimura, K. Tryka, R. Bagoutdinov, L. Hao, A. Kiang, J. Paschall, L. Phan, *et al.*, “The ncbi dbgap database of genotypes and phenotypes,” *Nature genetics*, vol. 39, no. 10, pp. 1181–1186, 2007.

[18] J. Kaye, C. Heeney, N. Hawkins, J. De Vries, and P. Boddington, “Data sharing in genomics–re-shaping scientific practice,” *Nature reviews. Genetics*, vol. 10, no. 5, p. 331, 2009.

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol–http/1.1,” Internet Engineering Task Force (IETF), Tech. Rep., 1999.

[20] J. Postel and J. Reynolds, “File transfer protocol,” 1985.

[21] S. Deorowicz and S. Grabowski, “Compression of dna sequence reads in fastq format,” *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.

[22] A. J. Cox, M. J. Bauer, T. Jakobi, and G. Rosone, “Large-scale compression of genomic sequence databases with the burrows–wheeler transform,” *Bioinformatics*, vol. 28, no. 11, pp. 1415–1419, 2012.

[23] S. W. Hodson, S. W. Poole, T. M. Ruwart, and B. W. Settlemyer, “Moving large data sets over high-performance long distance networks,” Citeseer, Tech. Rep., 2011.

[24] K. Holtman and A. Mutz, “Transparent content negotiation in http,” Internet Engineering Task Force (IETF), Tech. Rep., 1998.

[25] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, “Potential benefits of delta encoding and data compression for http,” in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 27, 1997, pp. 181–194.

[26] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[27] M. Aledhari and F. Saeed, “Design and implementation of network transfer protocol for big genomic data,” in *Big Data (BigData Congress), 2015 IEEE International Congress on*, IEEE, 2015, pp. 281–288.

[28] M. Aledhari, M. Hefeida, and F. Saeed, "Wired/wireless internet communications: 14th international conference, wwic 2016, thessaloniki, greece, may 25-27, 2016, revised selected papers," in. Springer International Publishing, 2016, ch. A Variable-Length Network Encoding Protocol for Big Genomic Data.

[29] S. Pichai and L. Upson, "Introducing the google chrome os," *The Official Google Blog*, vol. 7, 2009.

[30] Apple. (). Safari 3.1: Product overview, Apple.

[31] S. J. Davis and K. M. Murphy, "A competitive perspective on internet explorer," *American Economic Review*, pp. 184–187, 2000.

[32] M. Diaz, G. Juan, O Lucas, and A Ryuga, "Big data on the internet of things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 978–0.

[33] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[34] J. Postel, "Transmission control protocol," *The Internet Engineering Task Force*, 1981.

[35] A. Ng, P. Greenfield, and S. Chen, "A study of the impact of compression and binary encoding on soap performance," in *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005)*, Citeseer, 2005, pp. 46–56.

[36] B. A. Forouzan, *TCP/IP protocol suite*. McGraw-Hill, Inc., 2002.

[37] A. Bhushan, "File transfer protocol," *The Internet Engineering Task Force*, 1972.

[38] J. Touch, J. Heidemann, and K. Obraczka, "Analysis of http performance," *ISI Research Report ISI/RR-98-463,(original report dated Aug. 1996), USC/Information Sciences Institute*, 1998.

[39] M. Allman and S. Ostermann, "Ftp security considerations," *The Internet Engineering Task Force*, 1999.

[40] S. T. Redding, "Why ftp may forever be a security hole, and what you can do about it," *SANS SECURITY ESSENTIALS GSEC PRACTICAL ASSIGNMENT Version 1.2d*, 2002.

[41] C. Wilks, M. S. Cline, E. Weiler, M. Diehkans, B. Craft, C. Martin, D. Murphy, H. Pierce, J. Black, D. Nelson, *et al.*, "The cancer genomics hub (cghub): Overcoming cancer through the power of torrential data," *Database*, vol. 2014, 2014.

[42] B. Cohen, "Bittorrent-a new p2p app," *Yahoo eGroups*, 2001.

[43] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: The single-node case,” *IEEE/ACM transactions on networking*, vol. 1, no. 3, pp. 344–357, 1993.

[44] Y. Ren, J. Li, S. Shi, L. Li, G. Wang, and B. Zhang, “Congestion control in named data networking–a survey,” *Computer Communications*, vol. 86, pp. 1–11, 2016.

[45] 2017.

[46] Y. Rathore, M. K. Ahirwar, and R. Pandey, “A brief study of data compression algorithms,” *International Journal of Computer Science and Information Security*, vol. 11, no. 10, p. 86, 2013.

[47] L. P. Deutsch, “Deflate compressed data format specification version 1.3,” *The Internet Engineering Task Force*, 1996.

[48] J. Seward, “Bzip2 and libbzip2: A program and library for data compression,” *htpp://sources. redhat. com/bzip2*, 1998.

[49] L. J. Krakauer and L. Baxter, *Method of fixed-length binary encoding and decoding and apparatus for same*, US Patent 4,818,969, 1989.

[50] E. N. Gilbert and E. F. Moore, “Variable-length binary encodings,” *Bell System Technical Journal*, vol. 38, no. 4, pp. 933–967, 1959.

[51] S. Grumbach and F. Tahi, “A new challenge for compression algorithms: Genetic sequences,” *Information Processing &amp; Management*, vol. 30, no. 6, pp. 875–886, 1994.

[52] L. Chen, S. Lu, and J. Ram, “Compressed pattern matching in dna sequences,” in *Computational Systems Bioinformatics Conference, 2004. CSB 2004. Proceedings. 2004 IEEE*, IEEE, 2004, pp. 62–68.

[53] N. J. Larsson and A. Moffat, “Off-line dictionary-based compression,” *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1722–1732, 2000.

[54] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa, “A boyer—moore type algorithm for compressed pattern matching,” in *Annual Symposium on Combinatorial Pattern Matching*, Springer, 2000, pp. 181–194.

[55] D. Salomon and G. Motta, *Handbook of data compression*. Springer Science &amp; Business Media, 2010.

[56] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[57] J. Cleary and I. Witten, “Data compression using adaptive coding and partial string matching,” *IEEE transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.

[58] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, “A simple statistical algorithm for biological sequence compression,” in *Data Compression Conference, 2007. DCC’07*, IEEE, 2007, pp. 43–52.

[59] D. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[60] M. Effros, K. Visweswariah, S. R. Kulkarni, and S. Verdú, “Universal lossless source coding with the burrows wheeler transform,” *IEEE Transactions on Information Theory*, vol. 48, no. 5, pp. 1061–1081, 2002.

[61] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm,” 1994.

[62] M. Babaei, “A novel text and image encryption method based on chaos theory and dna computing,” *Natural computing*, vol. 12, no. 1, pp. 101–107, 2013.

[63] H. Wang, H. Fang, L. Xing, and M. Chen, “An integrated biometric-based security framework using wavelet-domain hmm in wireless body area networks (wban),” in *Communications (ICC), 2011 IEEE International Conference on*, IEEE, 2011, pp. 1–5.

[64] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta, “Pska: Usable and secure key agreement scheme for body area networks,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 1, pp. 60–68, 2010.

[65] Y. Liu, S. Tian, W. Hu, and C. Xing, “Design and statistical analysis of a new chaotic block cipher for wireless sensor networks,” *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 8, pp. 3267–3278, 2012.

[66] B. En-Jian, Z. Jun-Jie, and W. Liang-Cheng, “Wsn message authentication code based on chaos and xor-encryption,” *Sensors &amp; Transducers*, vol. 156, no. 9, p. 161, 2013.

[67] L. M. Adleman, “Molecular computation of solutions to combinatorial problems,” *Nature*, vol. 369, p. 40, 1994.

[68] Q. Zhang, L. Guo, and X. Wei, “Image encryption using dna addition combining with chaotic maps,” *Mathematical and Computer Modelling*, vol. 52, no. 11, pp. 2028–2035, 2010.

[69] M. Karakose and U. Cigdem, “Qpso-based adaptive dna computing algorithm,” *The Scientific World Journal*, vol. 2013, 2013.

[70] M. A. Mokhtar, S. N. Gobran, and E.-S. A. El-Badawy, “Colored image encryption algorithm using dna code and chaos theory,” in *Computer and Communication Engineering (ICCCE), 2014 International Conference on*, IEEE, 2014, pp. 12–15.

[71] N. H. UbaidurRahman, C. Balamurugan, and R. Mariappan, “A novel dna computing based encryption and decryption algorithm,” *Procedia Computer Science*, vol. 46, pp. 463–475, 2015.

[72] C. Gritti, W. Susilo, T. Plantard, and K. T. Win, “Privacy-preserving encryption scheme using dna parentage test,” *Theoretical Computer Science*, vol. 580, pp. 1–13, 2015.

[73] S. R. Moosavi, T. N. Gia, A.-M. Rahmani, E. Nigussie, S. Virtanen, J. Isoaho, and H. Tenhunen, “Sea: A secure and efficient authentication and authorization architecture for iot-based healthcare using smart gateways,” *Procedia Computer Science*, vol. 52, pp. 452–459, 2015.

[74] M. S. Farash, M. Turkanović, S. Kumari, and M. Hölbl, “An efficient user authentication and key agreement scheme for heterogeneous wireless sensor network tailored for the internet of things environment,” *Ad Hoc Networks*, vol. 36, pp. 152–176, 2016.

[75] J. S. Reis-Filho *et al.*, “Next-generation sequencing,” *Breast Cancer Res*, vol. 11, no. Suppl 3, S12, 2009.

[76] V. Marx, “Biology: The big challenges of big data,” *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[77] M. G. Langille and J. A. Eisen, “Biotorrents: A file sharing service for scientific data,” *PLoS One*, vol. 5, no. 4, e10071, 2010.

[78] U. Sangket, A. Phongdara, W. Chotigeat, D. Nathan, W.-Y. Kim, J. Bhak, C. Ngamphiw, S. Tongsima, A. M. Khan, H. Lin, *et al.*, “Automatic synchronization and distribution of biological databases and software over low-bandwidth networks among developing countries,” *Bioinformatics*, vol. 24, no. 2, pp. 299–301, 2008.

[79] M. Aledhari and F. Saeed, “Design and implementation of network transfer protocol for big genomic data,” *IEEE 4th International Congress on Big Data (BigData Congress 2015)*, 2015.

[80] Census, *World population*.

[81] P Deutsch, “Gzip file format specification version 4.3,” *The Internet Engineering Task Force*, 1996.

[82] A. J. Pinho and D. Pratas, “Mfcompress: A compression tool for fasta and multi-fasta data,” *Bioinformatics*, btt594, 2013.

[83] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[85] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[86] S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, IEEE, vol. 2, 2006, pp. 2169–2178.

[87] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson, “Multiple sequence alignment with the clustal series of programs,” *Nucleic acids research*, vol. 31, no. 13, pp. 3497–3500, 2003.

[88] R. G. Gallager, “Low-density parity-check codes,” *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21–28, 1962.

[89] Feb. 2016.

[90] Feb. 2016.

[91] M. L. Metzker, “Sequencing technologies—the next generation,” *Nature reviews genetics*, vol. 11, no. 1, pp. 31–46, 2010.

[92] *Illumina introduces the novaseq series—a new architecture designed to usher in the 100 usd genome*, 2017.

[93] . G. P. Consortium *et al.*, “A map of human genome variation from population-scale sequencing,” *Nature*, vol. 467, no. 7319, pp. 1061–1073, 2010.

[94] E. Pennisi, “Will computers crash genomics?” *Science*, vol. 331, no. 6018, pp. 666–668, 2011.

[95] T. J. Hudson, W. Anderson, A. Aretz, A. D. Barker, C. Bell, R. R. Bernabé, M. Bhan, F. Calvo, I. Eerola, D. S. Gerhard, *et al.*, “International network of cancer genome projects,” *Nature*, vol. 464, no. 7291, pp. 993–998, 2010.

[96] V. A. Fusaro, P. Patil, E. Gafni, D. P. Wall, and P. J. Tonellato, “Biomedical cloud computing with amazon web services,” *PLoS computational biology*, vol. 7, no. 8, e1002147, 2011.

[97] M. C. Schatz, B. Langmead, and S. L. Salzberg, “Cloud computing and the dna data race,” *Nature biotechnology*, vol. 28, no. 7, p. 691, 2010.

[98] L. D. Stein, “The case for cloud computing in genome informatics,” *Genome biology*, vol. 11, no. 5, p. 207, 2010.

[99] O. Trelles, P. Prins, M. Snir, and R. C. Jansen, “Big data, but are we ready?” *Nature Reviews Genetics*, vol. 12, no. 3, pp. 224–224, 2011.

[100] K. Kazimierczuk, A. Zawadzka, W. Koźmiński, and I. Zhukov, “Random sampling of evolution time space and fourier transform processing,” *Journal of biomolecular NMR*, vol. 36, no. 3, pp. 157–168, 2006.

[101] S. Gorn, R. W. Bemer, and J. Green, “American standard code for information interchange,” *Communications of the ACM*, vol. 6, no. 8, pp. 422–426, 1963.

[102] M. A. Jobling and P. Gill, “Encoded evidence: Dna in forensic analysis,” *Nature reviews. Genetics*, vol. 5, no. 10, p. 739, 2004.

[103] T. Lindahl, M. S. Satoh, G. G. Poirier, and A. Klungland, “Post-translational modification of poly (adp-ribose) polymerase induced by dna strand breaks,” *Trends in biochemical sciences*, vol. 20, no. 10, pp. 405–411, 1995.

[104] T. Tao, “Single letter codes for nucleotides,” *NCBI Learning Center. National Center for Biotechnology Information. Retrieved*, pp. 03–15, 2012.

[105] T. D. Blacker, “Fastq users manual version 1.2,” *Sandia National Laboratories, SAND88-1326*, 1988.

[106] A. C. Gilbert, S. Guha, P. Indyk, S Muthukrishnan, and M. Strauss, “Near-optimal sparse fourier representations via sampling,” in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ACM, 2002, pp. 152–161.

[107] E. J. Candès, J. Romberg, and T. Tao, “Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information,” *IEEE Transactions on information theory*, vol. 52, no. 2, pp. 489–509, 2006.

[108] M. Rudelson and R. Vershynin, “Sparse reconstruction by convex relaxation: Fourier and gaussian measurements,” in *Information Sciences and Systems, 2006 40th Annual Conference on*, IEEE, 2006, pp. 207–212.

[109] H. S. Shapiro and R. A. Silverman, “Alias-free sampling of random noise,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 225–248, 1960.

[110] F. J. Beutler and O. A. Leneman, "The spectral analysis of impulse processes," *Information and Control*, vol. 12, no. 3, pp. 236–258, 1968.

[111] O. A. Leneman and F. J. Beutler, "The theory of stationary point processes," *Acta Mathematica*, vol. 116, no. 1, pp. 159–190, 1966.

[112] F. J. Beutler and O. A. Leneman, "Random sampling of random processes: Stationary point processes," *Information and Control*, vol. 9, no. 4, pp. 325–346, 1966.

[113] O. A. Leneman, "Random sampling of random processes: Impulse processes," *Information and Control*, vol. 9, no. 4, pp. 347–363, 1966.

[114] E. Singh, "Sap hana platform for healthcare: Bringing the world closer to real-time personalized medicine," 2013.

[115] B. Néron, H. Ménager, C. Maufrais, N. Joly, J. Maupetit, S. Letort, S. Carrere, P. Tuffery, and C. Letondal, "Mobyle: A new full web bioinformatics framework," *Bioinformatics*, vol. 25, no. 22, pp. 3005–3011, Nov. 2009.

[116] J. C. Mogul and M. Nottingham, "Http header field registrations," 2005.

[117] B. (MD), *The NCBI Handbook*, 2nd. National Center for Biotechnology Information (US), 2013.

[118] M. E. BORDA, O. TORNEA, and T. HODOROGEA, "Secret writing by dna hybridization," *Acta Technica Napocensis-Electronica-Telecomunicatii (Electronics and Telecommunications)*, vol. 1, no. 50, pp. 21–24, 2009.

[119] R. Ye and W. Guo, "A chaos-based image encryption scheme using multi modal skew tent maps," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 4, no. 10, pp. 800–810, 2013.

[120] A. Burns, B. R. Greene, M. J. McGrath, T. J. O'Shea, B. Kuris, S. M. Ayer, F. Stroiescu, and V. Cionca, "Shimmer^{TM}–a wireless sensor platform for noninvasive biomedical research," *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1527–1534, 2010.

[121] J. D. Gibbons and S. Chakraborti, *Nonparametric statistical inference*. Springer, 2011.

[122] J. Thakur and N. Kumar, "Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis," *International journal of emerging technology and advanced engineering*, vol. 1, no. 2, pp. 6–12, 2011.